FPGA and Digital Signal Processing Laboratory Guide

1 Introduction

The laboratory consists of 9 labs. The goal of these labs is to give the students experience in developing simple DSP applications realized in FPGA devices. The labs make use of Memec's Spartan-3 development kit together with analog module containing fast analog-to-digital (ADC) and digital-to-analog (DAC) converters. The development kit is based on Xilinx XC3S400 chip derived from Spartan-3 FPGA family.

Xilinx Integrated Software Environment (ISE) is used by students to develop projects. The main design entries are VHDL project description files and Xilinx predefined building blocks named Intellectual Property (IP) cores. The software supports automatic synthesis, placement and routing of a design. A bit-file obtained as a result of these operations may be used to configure the Spartan FPGA device included in the development board. The JTAG programming cable attached to the PC parallel port makes it possible to download a bit-file directly into the FPGA device.

In the next sections a brief description of the Spartan-3 FPGA devices, the Memec development kit and the ISE software environment is given. Then the individual lab exercises are described. In the last section the main VHDL keywords and statements are briefly described.

2 Spartan-3 FPGA family overview

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) containing RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
- Input/Output Blocks (IOBs) controlling the flow of data between the I/O pins and the internal logic of the device.
- 18-Kbit dual-port RAM blocks providing data storage.

- Multiplier blocks which accept two 18-bit binary numbers as inputs and calculate the product.
- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

These elements are organized as shown in Figure 1. A ring of IOBs surrounds a regular array of CLBs. The XC3S400 device contains 896 CLBs, 288kbit embedded RAM, 16 dedicated multipliers and 4 DCMs.

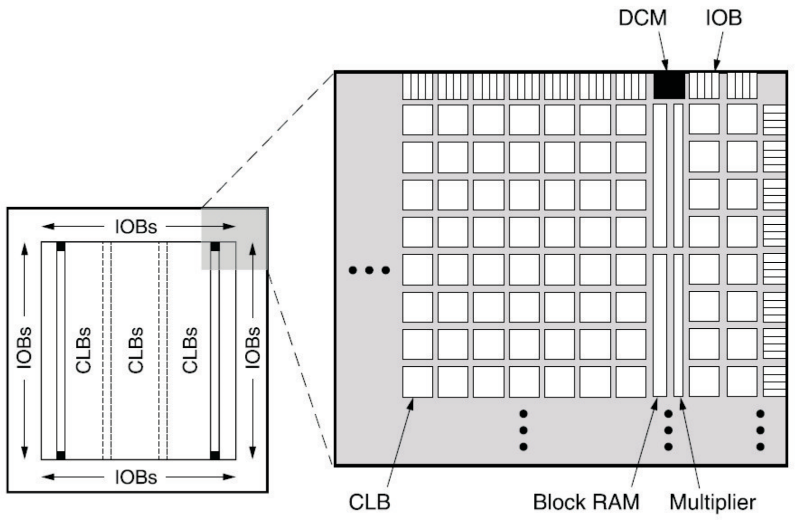


Figure 1: Spartan-3 family architecture

The main logic resource for implementing synchronous as well as combinatorial circuits is the Configurable Logic Block (CLB). Each CLB comprises four interconnected slices, as shown in Figure 2. These slices are grouped in pairs. Each pair is organized as a column with an independent fast carry chain. The carry chain supports implementing arithmetic functions such as addition.

All four slices have the following elements in common: two logic function generators (known as Look-Up Tables), two flip-flops, wide-function multiplexers, carry logic, and auxiliary arithmetic gates. The RAM-based Look-Up Table (LUT) is the main resource for implementing logic functions. Each of the two LUTs in a slice have four logic inputs and a single output. This permits any four-variable Boolean logic operation to be programmed into them. Wide-function multiplexers can be used to effectively combine

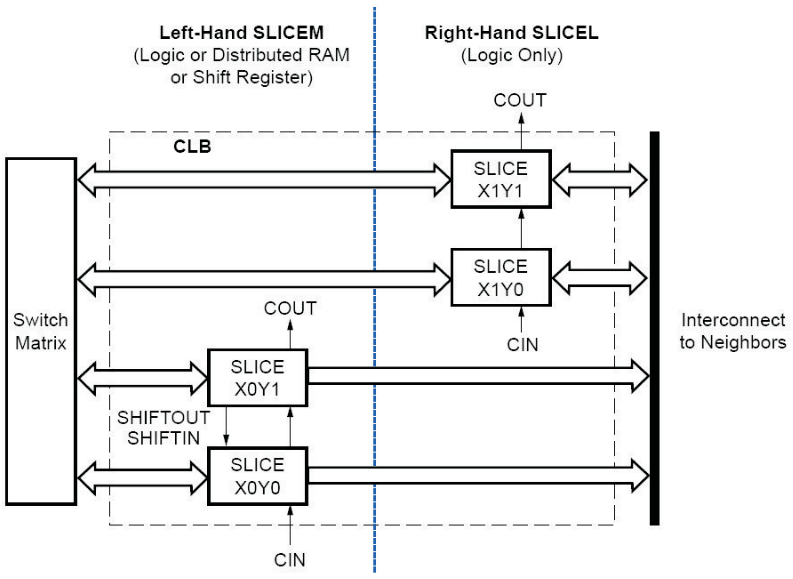


Figure 2: Configurable Logic Block

LUTs within the same CLB or across different CLBs, making logic functions with many more input variables possible.

Switch matrix (see Figure 2) allows programmable access into local and global routing resources. Clock signals are distributed by dedicated low-capacitance, low-skew network well suited to carrying high-frequency signals.

Spartan-3 FPGAs are programmed by loading configuration data into static memory cells that control all functional elements and routing resources. Before powering on the FPGA, configuration data is stored externally in a PROM or some other nonvolatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using one of the available modes, e.g. JTAG mode.

More detaile information about Spartan-3 FPGA devices is available in Data Sheet [3].

3 Memec Spartan-3 LC Development Board and P160 Analog Module

3.1 Memec Spartan-3 LC Development Board

The Spartan-3 LC Development Kit provides an easy-to-use evaluation platform for developing designs and applications based on the Xilinx Spartan-3 FPGA family. The development board utilizes the 400K-gate Xilinx Spartan-3 device (XC3S400-4PQ208CES). The board includes a 50 MHz clock, a user clock socket, 29 user I/O header pins, an RS-232 port, a USB 2.0 slave port, LEDs, switches, and additional user support circuits. The P160 Analog Module containing A/D and D/A converters is connected to the main board by dedicated 160 pin socket. This module is described briefly in the next subsection.

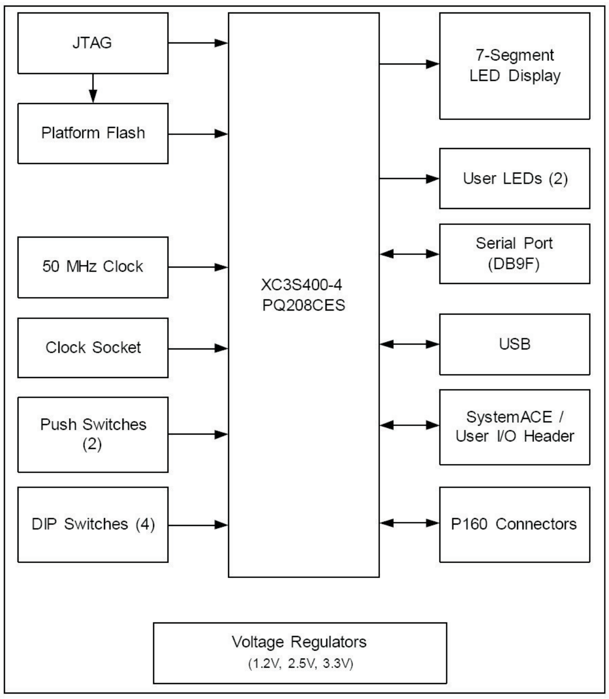


Figure 3: Spartan-3 LC Board block diagram

A simplified block diagram of the Spartan-3 LC development board is shown in Figure 3. Instructions for interfacing all included components are available in Memec Spartan-3 LC User's Guide [1]. In digital signal processing purposes, the most emphasis shall be put on using ADC and DAC components.

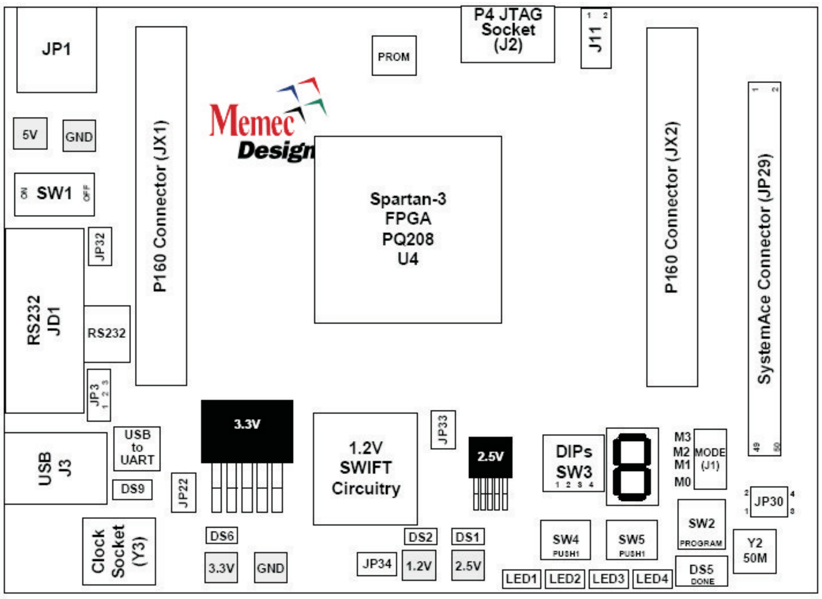
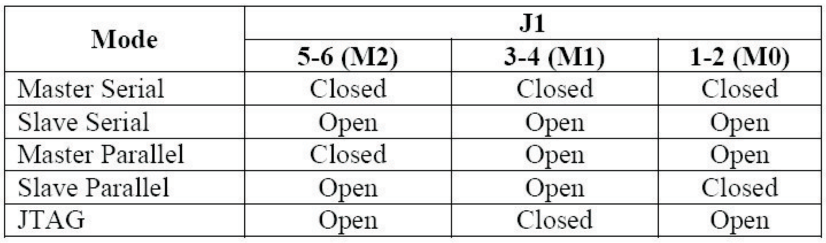


Figure 4: Spartan-3 LC Development Board

A JTAG connector (J2, see Figure 4) provides interface to the board's JTAG chain. This chain can be used to program the on-board ISP PROM and configure the Spartan-3 FPGA. The JTAG chain consists of an XCF02S Platform Flash PROM followed by an XC3S400 FPGA. The XCF02S Platform Flash In-System Programmable (ISP) PROM allows designers to store an FPGA configuration in nonvolatile memory. The JTAG port on the Platform Flash device is used to program the PROM with an `.mcs` file created by iMPACT in the Xilinx ISE software environment. Once the Flash has been programmed, the user can configure the Spartan-3 device by setting the Configuration Mode to Master Serial Mode (Jumper J1, see Figures 4 and 5). The Spartan-3 device configuration is initiated during power-up or by asserting the PROGAMn signal (by pressing the SW2 switch). The FPGA can be also configured directly through JTAG chain, without using of PROM. JTAG chain is connected into PC computer through dedicated Parallel cable.

3.2 P160 Analog Module

The Memec Design P160 Analog Module provides an advanced analog interface to compatible FPGA platforms. The Analog Module is comprised of four independent analog channels, two supporting analog inputs and two



Mode	J1		
	5-6 (M2)	3-4 (M1)	1-2 (M0)
Master Serial	Closed	Closed	Closed
Slave Serial	Open	Open	Open
Master Parallel	Closed	Open	Open
Slave Parallel	Open	Open	Closed
JTAG	Open	Closed	Open

Figure 5: FPGA Configuration Mode Select

supporting analog outputs. Both channels are identical. The Texas Instruments ADS807 12-bit, 53Msps A/D converters are used to convert incoming analog signals into 12-bit data for the FPGA located on the baseboard. Analog outputs can be generated using the two DAC902 12-bit, 165Msps D/A converters from Texas Instruments. Gain and filtering is provided on the D/A outputs. Control of the ADCs and DACs is handled by the FPGA through the P160 digital interface.

3.2.1 D/A Converters

The FPGA interfaces to the DACs through 12-bit registers, which add a clock cycle delay between data out from the FPGA and the DAC analog outputs. Two independent data channels, one channel for each DAC, are driven from the FPGA. The DACs interface signals are:

- 12 bits of input data. Output voltage values corresponding to input binary values are shown below.

111111111111	$3.5V$
100000000000	$2.5V$
000000000000	$1.5V$

  As can be easily seen, the DAC output is normally DC coupled for a 2 V peak-to-peak (Vp-p) signal centered at 2.5 V.

- Two clock signals: DAC Clock (CLK) and Register Clock (CLK2), rising edge active. The CLK2 signal latches the digital DAC data from the FPGA into the register. The CLK signal latches the output data from the register into the DAC. On the falling edge of the CLK signal, the DAC output changes to the newly latched value.

- Reference Select (RefSel) control signal, which makes possible to disable internal reference voltage source and to use external reference input (Low = Internal, High = External Reference).

- Power Down (PD) control signal (Low = Normal, High = Power Down Mode).

3.2.2 A/D converters

Texas Instruments ADS807 converters provide 12-bit resolution at up to 53 Msps. The digital data out of the A/Ds is latched into external buffers and then passed to the FPGA through the P160 interface. The range of the input voltage is dependent on the Full Scale Select control signal to the A/D. Before conversion the input signal is AC coupled, biased to 2.5 volts for unipolar operation, and buffered through the op amp.The ADCs interface signals are:

- 12 bits of output data (binary range "000000000000" to "111111111111").
- Full Scale Select control signal (FsSel). Setting this signal to a logic high allows a 1.5 Vp-p input to the board. Setting the Full Scale Select to low, selects a 1 Vp-p input range to the board (i.e. $-0.5V$ voltage corresponds to "000000000000" value and $+0.5V$ corresponds to "111111111111" value).
- Reference Select (RefSel) control signal (Low = Internal Reference, High = External Reference).
- Output Enable (OE) control signal (Low = Output Enabled, High = Tri-Stated outputs).
- Convert clock (CLK) signal. The ADS807 samples the input signal on the rising edge of the CLK input. Output data values are valid at the outputs 6 clock cycles later, after the rising edge of the clock.

More detailed information about the Analog Module is available in [2].

4 Xilinx Integrated Software Environment (ISE)

The Xilinx Integrated Software Environment (ISE) is a complex set of tools to design programmable logic devices projects. ISE tools allow the design to be entered several ways including graphical schematics, state machine diagrams, VHDL, Verilog and Intelectual Property (IP) cores. This guide will focus on VHDL entry, but the other methods are similar and can be easily explored once the reader is comfortable with the ISE software.

The primary ISE tools used in this laboratory are:

- ISE Project Navigator which integrates all ISE tools and provides windows displaying project hierarchy and the design process, as well as a context-sensitive HDL Editor.
- Core Generator module which offers an optimized, predefined set of building blocks for common functions - simplifying design steps and bringing the design to completion faster.
- iMPACT tool that allows to configure FPGA devices through JTAG programming cable.
- ModelTechnology ModelSim that enables functional simulation of a VHDL design.

4.1 HDL design flow

The process of converting hardware description language (HDL) files into a configuration bitstream which can be used to program the FPGA, is done in several steps.

First, the HDL files are synthesized. Synthesis is the process of converting behavioral HDL descriptions into a network of logic gates. The synthesis engine takes as input the HDL design files and a library of primitives. Primitives are not necessarily just simple logic gates like AND and OR gates and D-registers, but can also include more complicated things such as shift registers and arithmetic units. Primitives also include specialized circuits such as DCMs (Digital Clock Manager) that cannot be inferred by behavioral HDL code and must be explicitly instantiated. The libraries guide in the Xilinx documentation provides an complete description of every primitive available in the Xilinx library. There are occasions when it is helpful or even necessary to explicitly instantiate primitives, but it is much better design practice to write behavioral code whenever possible.

Synthesis process is supported by Xilinx engine known as XST. XST takes as input a verilog or VHDL file and generates a .ngc file. A synthesis report file (.srp) is also generated, which describes the logic inferred for each part of the HDL file, and often includes helpful warning messages.

The .ngc file is then converted to an .ngd file. This step mostly seems to be necessary to accommodate different design entry methods, such as third-part synthesis tools or direct schematic entry. Whatever the design entry method, the result is an .ngd file.

The .ngd file is essentially a netlist of primitive gates, which could be implemented on any one of a number of types of FPGA devices Xilinx manufacturers. The next step is to map the primitives onto the types of resources (logic cells, i/o cells, etc.) available in the specific FPGA being targeted. The output of the Xilinx map tool is an .ncd file.

The design is then placed and routed, meaning that the resources described in the .ncd file are then assigned specific locations on the FPGA, and the connections between the resources are mapped into the FPGAs interconnect network. The delays associated with interconnect on a large FPGA can be quite significant, so the place and route process has a large impact on the speed of the design. The place and route engine attempts to honor timing constraints that have been added to the design, but if the constraints are too tight, the engine will give up and generate an implementation that is functional, but not capable of operating as fast as desired.

The output of the place and route engine is an updated .ncd file, which contains all the information necessary to implement the design on the chosen FPGA. All that remains is to translate the .ncd file into a configuration bitstream in the format recognized by the FPGA programming tools. Then the programmer is used to download the design into the FPGA, or write the appropriate files to a flash PROM, which is then used to configure the FPGA.

4.2 Constraints

Project implementation information such as device pin allocations and pin electrical properties are usually not stored in HDL source files, they are stored in separate constraints files. Constraints specify placement, implementation, naming, signal direction, and timing considerations for timing analysis and for design implementation. Constraints can be defined by manually editing ascii file (with extension `.ucf` - User Constraint File) or defined using Pinout and Area Constraints Editor (PACE) application as well as Constraint Editor application.

A `.ucf` file is simply a list of constraints, such as

```
NET "Dip<3>" LOC = "P26";
```

which indicates that bit 3 of the signal `Dip` (which should be a port in the top-level HDL module) should be assigned to pin P26 on the FPGA. Sometimes it is useful to combine several related constraints on one line, using "`|`" characters to separate constraints.

```
NET "Dip<3>" loc="P26" | FAST | IOSTANDARD=LVDCI_33 | DRIVE=12;
```

The above example again assigns bit 3 of the signal `Dip` to pin P26, and also specifies that the i/o driver should be configured for fast slew rate, 3.3V LVTTL level signaling (with a built-in series termination resistor), and a drive strength of 12mA. All of the necessary pin constraints for the Memec Development Board have been written for you in a template UCF file.

4.3 Core Generator

The Xilinx CORE Generator System provides a catalog of user-customizable blocks ranging in complexity from simple arithmetic operators (adders, accumulators, and multipliers), memories and FIFOs, to networking interfaces and DSP building blocks such as filters and transforms.

You can start the CORE Generator from the ISE environment or directly from Windows environment. Before starting the CORE Generator, an ISE project should be created, so generated cores will be added to it.

For each core it generates, the CORE Generator System produces an Electronic Data Interchange Format (`.edn`) netlist file (EDN file), a VHDL (or Verilog) template file (`.vho`) and a VHDL (Verilog) wrapper file (`.vhd`). It may also create one or more NGC and NDF files. NGC files are produced for certain cores only.

The Electronic Data Netlist (EDN) and NGC files contain the information required to implement the module in a Xilinx FPGA. VHD wrapper file is provided to support functional simulation. VHO template file contains code that can be used as a template for instantiating a core in a VHDL design. The only thing to do is to copy and paste component declaration and instantiation from this VHO file into your design (renaming the signals if needed) and add EDN netlist into project.

4.4 iMPACT

Xilinx iMPACT, which can be started from ISE or directly from operating system, enables you to configure PLD designs through four modes of configuration, among which is JTAG mode that is used for the Development Board. Using JTAG programming cable you can configure FPGA and/or store configuration in the flash PROM.

Before downloading configuration into XCF02S Platform Flash PROM, an appropriate PROM configuration bit-file have to be created. Such file (with extension `.mcs`) can be created in iMPACT application (File Generation Mode).

5 Laboratory exercises

5.1 Lab 1: Introduction to ISE software and Memec Spartan-3 Development Kit. Interfacing A/D and D/A converters.

The purpose of the first Lab is to give the students opportunity to familiarize with the tools used in the laboratory.

A simple ISE project template suitable for use with the Development Kit is available for you. The template consists of project file `Template.npl`, VHDL source file `Templatetop.vhd` and user constraints file `Memec_kit.ucf`. The VHDL source file contains a top level entity declaration and its architecture body definition. The entity declaration is as follows:

```
entity Templatetop is
    Port (
        Clk_in : in std_logic; --50MHz clock input
        Dac1 : out std_logic_vector(11 downto 0); --DAC 1 data
        Dac1_PD, Dac1_Clk, Dac1_Clk2, Dac1_RefSel : out std_logic;
        --DAC 1 control signals
        Dac2 : out std_logic_vector(11 downto 0); --DAC 2 data
        Dac2_PD, Dac2_Clk, Dac2_Clk2, Dac2_RefSel : out std_logic;
        --DAC 2 control signals
        Adc1 : in std_logic_vector(11 downto 0); --ADC 1 data
        Adc1_fssel, Adc1_refsel, Adc1_oe, Adc1_clk : out std_logic;
        --ADC 1 control signals
        Adc2 : in std_logic_vector(11 downto 0); --ADC 2 data
        Adc2_fssel, Adc2_refsel, Adc2_oe, Adc2_clk : out std_logic;
        --ADC 2 control signals
        Push1, Push2 : in std_logic; --Pushbuttons
        Dip : in std_logic_vector(3 downto 0); --DIP switch
        Led : out std_logic_vector(4 downto 1); --LEDs
        Display : out std_logic_vector(6 downto 0); --LED Display
        USB_RESET, USB_DSR, USB_CTS,
            USB_DCD, USB_RI, USB_Tx : out std_logic;
        USB_DTR, USB_RTS, USB_Rx : in std_logic
        --USB port data and control signals
    );
end Templatetop;
```

Unused port declarations shall be removed when designing particular projects, as well as the UCF file should be edited to remove constraints for unused pins.

Remember that port names in the top level entity and in the UCF file have to be identical.

The architecture body contains description of the clock distribution network making use of the DCM (Digital Clock Manager) and simple port signals assignments to set appropriate DAC and ADC control signals as well as to turn the LED and Display lights off.

The clock distribution network consists of DCM and buffers for incoming clock signal and for output clock to DAC and ADC (`BUFG` and `OBUF` components). The primary task for the DCM is to provide suitably phase shifted clock for DAC clocking. The DAC data output from FPGA is changing synchronously to main FPGA clock signal, thus the DAC clock signal have to be phase shifted to avoid signal glitches in the DAC analog output. The 90° shifted clock signal (from `Clk90` output of the DCM) is provided to the DAC.

Preparations

- Read carefully previous sections of this Laboratory Guide.

Assignments

- Start ISE Project Navigator. Create a new project with HDL top-level module type. Select Spartan3 device family, XC3S400 device, pq208 package. Add a new VHDL source file with entity containing one input port `Push1` and one output `Led1`. Create a new text file, save it with `.ucf` extension and add it to the project. Write the following constraints in the UCF file:

```
NET "Push1"  LOC = "P22" | PULLUP | IOSTANDARD = LVCMOS33;
NET "Led1"   LOC = "P19" | FAST | DRIVE = 24 |
                           IOSTANDARD = LVCMOS33;
```

  In the architecture body, write a single line that allows to switch on and off the LED1 by pushing the Pushbutton on the Board (`Led1 <= Push1;`). Compile the project by double clicking on Generate Programming File in Processes for Source window.

  After successful programming file generation, download the Bitstream to the FPGA on the board. Make sure that JTAG cable is properly connected to the Board and start iMPACT application. Select options: Boundary-Scan Mode, then Automatically identify Boundary-Scan chain. A window with JTAG chain similar to that shown in

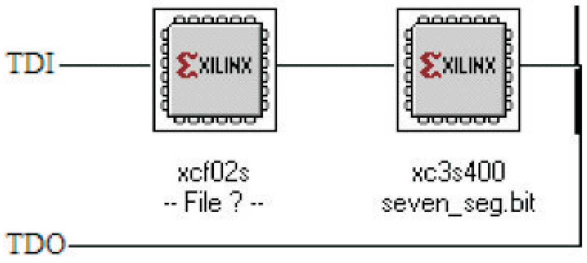


Figure 6: JTAG chain

figure 6 should appear. In the Assign New Configuration File dialog, click Cancel to skip the PROM, then select appropriate bitfile for the FPGA. Right click on the FPGA and select Program. After a while the FPGA device is programmed.

- Open the template project `Template.npl` in the Project Navigator. Familiarize with the VHDL source file and the User Constraints File. Pay attention to the port and signal names in the source file and UCF file. Main 50MHz FPGA Clock is `Clk` signal. Analyze the clock components connections.

- Add a single process to the architecture body that is activated on rising edge of `Clk`. Write statements that will be simply copying input signal from ADC1 to DAC1 output. Compile the design, download to the Development Board and test using function generator and oscilloscope.

- Modify the code so that the output signal will be rectified version of the input signal.

5.2 Lab 2: Simple signal generation (square, triangle, sawtooth).

The purpose of this lab is to design simple linear signals generator.

Assignments

- Create a new project for Spartan XC3S400 device. Copy source code from `Templatetop.vhd` and add to project a copy of UCF file `Memec_kit.ucf`. Remove all ports except of DAC1 data and control signals ports.
- Modify the project to generate square wave on the DAC1 output. Required frequency of the signal will be given to students during lab.
- Modify the project to allow changing frequency by switching pushbuttons, i.e. switching one pushbutton should increase frequency and the other – decrease.
- Add ports to interface second DAC (DAC2) to the top entity. Generate the sawtooth wave on the second DAC output. To achieve this, a simple counter should be created and its output connected to the DAC2 data input. Allow changing frequency of the sawtooth wave.
- Design the triangle wave generator and similarly allow changing frequency.

5.3 Lab 3: Sine wave generation by Look-Up Table and CORDIC algorithm.

In this lab the students are required to design a sine wave generator. Two methods to accomplish this task will be tested. The first is using Sine/Cosine Look-Up Table core and the second – using CORDIC algorithm core. The Xilinx Core Generator supports making use of these cores.

Preparations

- Read the Sine/Cosine Look-Up Table core specification.
- Read the CORDIC core specification.

Assignments

- After creating a new ISE project, start Core Generator system. From the list of available cores, find the Sine/Cosine Look-Up Table. Specification of each core is available after right clicking the core name in the list. Open datasheet with Sine/Cosine Look-Up Table specification and read through it.
- Generate the Sine/Cosine Look-Up Table core. Add it to the ISE project and copy from `.vho` file the component declaration and instantiation templates into your design.
- To generate sine wave, the Look-Up Table input should be fed by constantly growing argument (from 0 to 2π and so forth). The core computes $\sin(\theta)$ where

  $$\theta = \mathrm{THETA}\frac{2\pi}{2^{\mathrm{THETA\_WIDTH}}}[rad]$$

  where THETA is an integer input angle. Outputs sine and cosine are expressed as fractional fixed-point values. See specification of the core for more details. Generate sine wave of given frequency and observe it on the oscilloscope. Note that the DAC input value is in an unsigned format, thus simple conversion of Look-Up Table output have to be done.
- Create a new project. Generate CORDIC core in the Core Generator. The CORDIC core implements a generalized coordinate rotational digital computer (CORDIC) algorithm, to iteratively solve trigonometric, hyperbolic and square root equations. More details about this algorithm is available in the CORDIC core specification. Generate sine wave using this core.

- Compare resources occupied by generators designed by both methods. The number of used CLBs, LUTs, memory blocks and multiplier blocks can be read from Map Report file as well as Place & Route Report file.

5.4 Lab 4: Amplitude modulation: AM, DSB-SC.

The purpose of this lab is to design simple amplitude modulators: AM and DSB-SC (Double SideBand Suppressed Carrier).

Preparations

- Read the Multiplier core specification.

Assignments

- Design DSB-SC modulator. The modulating signal from ADC1 input should modulate amplitude of the carrier signal. The carrier sinusoidal signal is generated inside the FPGA. Thus the design should consist of sine generator (similar to that created in the previous lab exercise) and multiplier that calculates the product of the carrier and modulating signal from ADC. The carrier frequency will be given during the lab. The multiplier can be designed using Multiplier core or directly instantiated – Xilinx library includes `MULT18X18` and `MULT18X18S` elements that represents embedded multiplier blocks in the FPGA (see Spartan-3 supported design elements [4]). However, using of the Multiplier core is more flexible. Create DSB-SC modulator using both methods of forming multiplier.
- Design AM modulator. Since the AM signal is DSB plus carrier, it can be generated by simply adding synchronous carrier to the DSB signal.
- Observe the AM signal on the oscilloscope when the modulating signal is sine with $1V$ peak-to-peak. Calculate the modulation index of the signal. Change the project to obtain modulation index 0.5.

5.5 Lab 5: FIR filter design using Distributed Arithmetic FIR core.

In this lab a Finite Impulse Response filter will be realized by students. The Distributed Arithmetic FIR Filter core will be used.

Distributed arithmetic approach employs no explicit multipliers in the design, only look-up tables (LUTs), shift registers, and a scaling accumulator. A simplified view of a DA FIR is shown in Figure 7. In its most obvious and direct form, DA-based computations are bit-serial in nature. Extensions to the basic algorithm remove this potential throughput limitation. The advantage of a distributed arithmetic approach is its efficiency of mechanization. The basic operations required are a sequence of table look-ups, additions, subtractions and shifts of the input data sequence. All of these functions efficiently map to FPGAs. Input samples are presented to the input parallel-to-serial shift register (PSC) at the input signal sample rate. As the new sample is serialized, the bit-wide output is presented to a bit-serial shift register or time-skew buffer (TSB). The TSB stores the input sample history in a bit-serial format and is used in forming the required inner-product computation. The TSB is itself constructed using a cascade of shorter bit–serial shift registers. The nodes in the cascade connection of TSBs are used as address inputs to a look-up table. This LUT stores all possible partial products over the filter coefficient space.

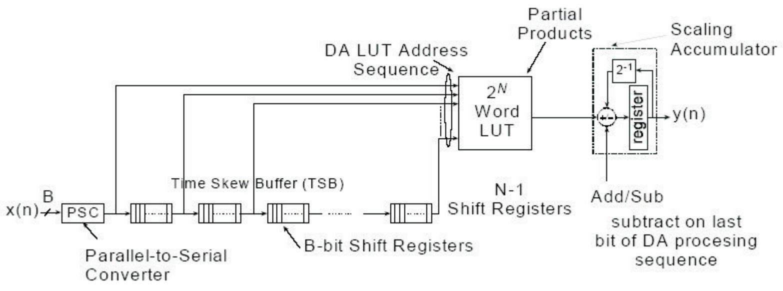


Figure 7: Serial Distributed arithmetic FIR Filter

Another available from Xilinx FIR filter core is the MAC FIR core. It uses one or more time-shared multiply accumulate (MAC) functional units to service the N sum-of-product calculations in the filter. This approach is similar to Digital Signal Processors solutions except that in the FPGA core a lot of MAC units can be working in parallel improving filter throughput.

Preparations

- Read the Distributed Arithmetic Fir Filter core documentation.
- Calculate the filter coefficients from given in advance filter specification. This calculation could be done for example using Matlab Filter Design & Analysis Tool.
- Write the calculated coefficients in the Xilinx `.coe` file format. This is an ASCII text file with a single-line header that defines the radix (base-2, base-10, or base-16) of the number representation used for the coefficient data, followed by the coefficient values themselves. An example is shown below for an 8-tap filter.

  ```
  radix=10;
  coefdata=20, -256, 200, 255,39, 117, -235, 47;
  ```

  The coefficient values may also be placed on separate lines.

  When using Matlab FDA Tool, the `.coe` file can be generated automatically.

Assignments

- Create a new project, making use of the Template project. Generate and instantiate the Distributed Arithmetic FIR Filter core using previously written `.coe` coefficient file and selecting parallel implementation option.
- Compile the design and download to the Development Board. Check how it works using function generator and oscilloscope.
- To allow frequency response measurement by spectrum analyzer, change filter clock frequency to $100kHz$. This requires designing appropriate clock frequency divider.
- Measure the frequency response of filter using HP35665 spectrum analyzer. Save data into floppy disc and open the file in the PC computer. The data can be read by `Viewdata.exe` application as well as convert into ascii format by `Sdftoasc.exe` application. More details will be given during lab.

- Regenerate the DA FIR Filter core selecting serial implementation. Compare the resources (CLBs, LUTs, RAM and multiplier blocks) occupied by both filter realizations. What is the maximum sampling frequency in these cases?

Report

- Create report including magnitude and phase response of the designed filter measured by spectrum analyzer.

5.6 Lab 6: Demodulation of AM signal.

The purpose of this lab is to design AM demodulator. The simplest way to realize AM demodulation is an envelope detector that tracks the peaks of the signal waveform. Envelope detection can be realized as rectification and then low pass filtration of AM signal.

Preparations

- Calculate coefficients of an appropriate low pass FIR filter for envelope detection. The filter shall be used for the AM signal detection (sampling frequency 50MHz) with maximum frequency of modulating signal 20kHz and carrier frequency 2MHz. The filter should have maximum 3dB ripple in the modulating signal band (0 – 20kHz) and minimum 40dB attenuation of carrier frequency signal.

Assignments

- Design envelope detector consisting of rectifier and low pass FIR filter meeting specifications given above. Use Distributed Arithmetic FIR Filter core with full parallel realization. Carrier frequency should equal 2MHz.
- In this lab, two Development Boards shall be used. Download into the first Board configuration of AM modulator created during the 4th lab. Then, download into the second board configuration of AM demodulator. Connect input of the modulator to the function generator and observe simultaneously signals on the input and the output of the demodulator. Try different types and frequencies of modulating signal.

5.7 Lab 7: Fast Fourier Transform (FFT)

The goal of this lab is to implement the Fast Fourier Transform calculation. To achieve it Xilinx FFT core is used. The FFT core computes an N-point forward DFT or inverse DFT (IDFT) where N can be 2^m, $m = 3–16$.

Preparations

- Read the Fast Fourier Transform core documentation
- Read the Using Block RAM in Spartan-3 FPGAs application note from Xilinx

Assignments

- Create a new project. Generate a 256-point FFT core.
- Design project for calculating a 256-point FFT of input from ADC1 signal and storing results in a Block RAM. FFT shall be calculated after pushing PUSH1 button. Dual port block RAM should be used: one port for writing FFT and another for reading data.
- FFT results shall be displayed on the oscilloscope. Thus in one of the DAC outputs a periodically repeated signal representing the calculated FFT should be generated. In the other analog output, a pulses for synchronizing the oscilloscope should be generated.
- Observe spectrum of the basic signals: sine, rectangular, triangle and AM.
- Change width of the FFT to 1024 points.

5.8 Lab 8: IIR filter design.

In this lab an Infinite Impulse Response (IIR) filter shall be realized. Because IIR filters are very sensitive to quantization errors, they are usually implemented as cascade (or parallel) coupled second order sections. A single second order section description in VHDL have been written for you. This section is as shown in figure 8.

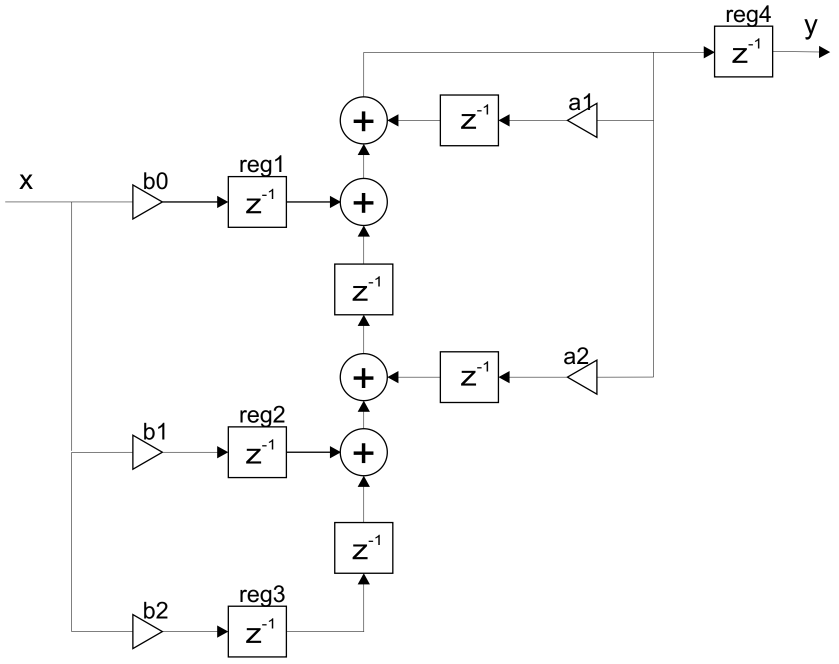


Figure 8: Second order section

Registers reg1...reg4 realize pipelining – they do not affect amplitude response of section, but they increase maximum operating frequency (throughput). Multipliers are composed of adders and shifters. A method for replacing a constant coefficient multiplier with shifters and adders is based on canonic signed digit (CSD) expression of multiplicand. This expression is a sum of fewest signed power–of–two terms:

$$c = \sum_{r=0}^{R} s(r)2^r, \quad s(r) = -1, 0, 1 \tag{1}$$

For example, 59 can be expressed as $59 = 2^6 - 2^2 - 2^0$ whose CSD expression is $1000\overline{1}0\overline{1}$ where the overline stands for negative digits. The number of adders (subtractors) required to implement the multiplier is equal to the number of nonzero signed digits minus one. Multiplication with power-of-two (e.g. 2^6) is simply shift operation and requires no hardware. Thus the multiplication with 59 requires two adders (in fact subtractors which have the same implementation cost as adders). Example multiplier that performs multiplication with 1.8125 is shown in figure 9.

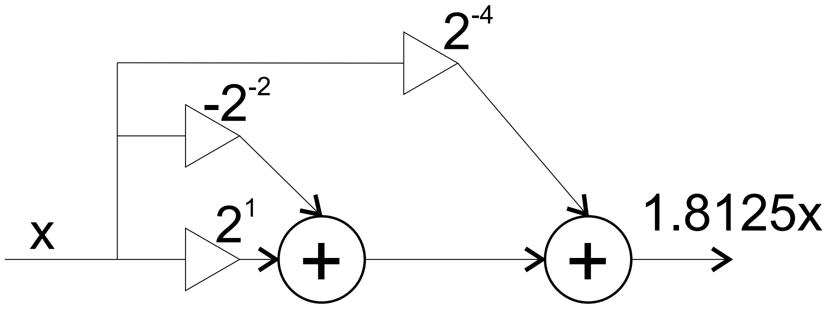


Figure 9: Example constant coefficient multiplier

Multiplier for CSD representation with 3 nonzero digits have been implemented in `multsdc3.vhd` source file. Second order section entity `biqiir` have been implemented in `biqiir.vhd` source file. The entity declaration is as follows:

```
entity biqiir is
    Generic ( b0 : coeff := (-1,-2,-3); --coefficients
            b1 : coeff := (-1,-2,-3);
            b2 : coeff := (-1,-2,-3);
            a1 : coeff := (-1,-2,-3);
            a2 : coeff := (-1,-2,-3);
            sb0 : std_logic_vector(1 to 3) := "000"; --signs
            sb1 : std_logic_vector(1 to 3) := "000";
            sb2 : std_logic_vector(1 to 3) := "000";
            sa1 : std_logic_vector(1 to 3) := "000";
            sa2 : std_logic_vector(1 to 3) := "000";
            length : integer --word length
            );
    Port ( clk : in std_logic;
           rst : in std_logic;
           x : in std_logic_vector(length-1 downto 0);
           y : out std_logic_vector(length-1 downto 0));
end biqiir;
```

Type `coeff` – declared in `iir_pack.vhd` package – is 3-element integer table for exponents of nonzero digits storing. Signs of this digits (1 means "minus", 0 – "plus") are stored in 3-element `std_logic_vector` (`sb0`...`sa1` generics).

Preparations

- Calculate coefficients of IIR filter that meets specifications given in advance. Factorize obtained transfer function into the cascade second order sections form.
- Quantize all coefficients of second order sections into the nearest CSD representation with 3 nonzero digits. It can be done using Matlab `sptconv.m` script that is available for students. Write down exponents and signs of nonzero digits.

Assignments

- Create a new ISE project. Add to the project: `iir_pack.vhd` package file, `multsdc3.vhd` multiplier source file and `biqiir.vhd` second order section source file.
- Create a new VHDL source for the top level entity description. Declare `biqiir` component and instantiate appropriate number of sections for the filter. Coefficients shall be declared as generics.
- Compile the project and download into the board. Test if it works using sine generator and oscilloscope.
- To allow frequency response measurement by spectrum analyzer, change filter clock frequency to $100kHz$.
- Measure the frequency response of filter using spectrum analyzer. Save data into floppy disc.

Report

- Create report including magnitude and phase response of the designed filter measured by spectrum analyzer.

5.9 Lab 9: Simple digital modulation techniques: binary FSK and PSK.

During this lab, a FSK and PSK modulator shall be created.

Binary FSK (Frequency Shift Keying) scheme uses two different carrier frequencies to represent 1 and 0 symbol:

$$\begin{aligned} s_1(t) &= A\cos(2\pi f_1 t) \\ s_2(t) &= A\cos(2\pi f_2 t) \end{aligned}$$

Coherent FSK modulator block diagram is shown in figure 10. To achieve continuous phase (no signal discontinuity), the frequencies should meet:

$$\begin{aligned} f_1 &= \frac{2n+m}{4T} \\ f_2 &= \frac{2n-m}{4T} \end{aligned}$$

where n, m are integers and T is the bit period of the binary data.

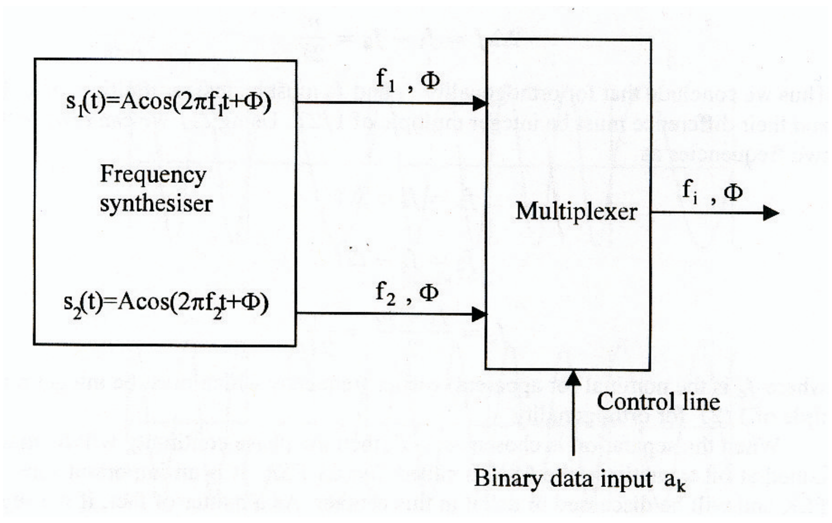


Figure 10: FSK modulator

In binary PSK (Phase Shift Keying) data are represented by two signals with different phases. Typically these phases are 0 and π, so the signals are:

$$\begin{aligned} s_1(t) &= A\cos(2\pi f_c t) \\ s_2(t) &= -A\cos(2\pi f_c t) \end{aligned}$$

One signal represents 0 and the other – 1.

Assignments

- Design binary FSK modulator. Carrier frequencies should meet relationships given above. The bit period will be given during the lab.
- Modulator shall send periodically 4 bits that are set on the DipSwitch on the development board. Observe the FSK signal and its spectrum on the scope.
- Design binary PSK modulator that periodically sends 4 bits. Observe the generated signal.

References

[1] *Memec Spartan-3 LC User's Guide*

[2] *P160 Analog Module User Guide*

[3] *Spartan-3 FPGA Family: Complete Data Sheet*

[4] *Spartan-3 supported design elements*, `http://toolbox.xilinx.com/docsan/xilinx6/books/data/docs/lib/lib0022_8.html`

[5] *Using Block RAM in Spartan-3 FPGAs*

[6] *Using Embedded Multipliers in Spartan-3 FPGAs*

[7] *Using the ISE Design Tools for Spartan-3 FPGAs*

[8] *Using Spartan-3 IP Cores*

[9] *VHDL language guide*, `http://www.acc-eda.com/vhdlref/`

[10] P.J. Ashenden *The VHDL Cookbook*

References are available on `http://alfa.iele.polsl.gliwice.pl/~wojsu/`