Digital Signal Processing on FPGA devices
Q: DSP on FPGA?
A: Matlab ↔ HDL cosimulation & HLS

- Mathworks
- Xilinx – System Generator for DSP,
- High Level Synthesis
- Aldec AHDL MATLAB
References

- www.mathworks.com
- www.xilinx.com
- www.aldec.com
- www.synopsys.com

- www.dsp-fpga.com
- www.edacafe.com
- www.eetimes.com
- http://www.dvcon.org/
- http://www.deepchip.com/
- http://www.demosondemand.com
DSP platforms

Question?

From DSPmagazine

Forward Concepts 300 DSP professionals from 30 countries, "Which chip types are employed for DSP algorithm execution (rather than data processing) in your applications"
DSP on FPGA

FPGA-based DSP - Parallelism

\[
\begin{align*}
\frac{600 \text{ MHz}}{1 \text{ clock cycle}} &= 600 \text{ MSPS} \\
\frac{250 \text{ MHz}}{1 \text{ clock cycle}} &= 250 \text{ MSPS}
\end{align*}
\]

Conventional DSP Processor - Serial

\[
\frac{1 \text{ GHz}}{126 \text{ clock cycles}} = 8 \text{ MSPS / MAC unit}
\]

Massive Parallel Processing

J.Kasperek & P.J.Rajda © 2015 Katedra Elektroniki AGH
# Digital Signal Processor Selection

DSPs bring computing performance, real-time processing, and power efficiency to diverse applications ranging from sensors to servers. Our product range spans high-performance real-time needs, to power-efficient processors with industry-leading lowest active power needs. Choose one of our scalable solutions below.

- Select the right DSP

<table>
<thead>
<tr>
<th></th>
<th>Ultra-Low Power C55x</th>
<th>Single Core C674x</th>
<th>Multicore C66x</th>
<th>Multicore ARM + DSP 66AK2x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed (MHz)</td>
<td>50-300MHz</td>
<td>200-456MHz</td>
<td>850MHz-1.4GHz</td>
<td>ARM: 1.25 GHz to 1.4 GHz; DSP: 1.25 GHz to 1.4 GHz</td>
</tr>
<tr>
<td>FLOPS/MACs/MIPS</td>
<td>Up to 600 MMACS</td>
<td>Up to 2.7 GFLOPS</td>
<td>Up to 179 GFLOPS</td>
<td>Up to 19600 DMIPS; Up to 67.2 GFLOPS</td>
</tr>
<tr>
<td>Power Consumption (typ.)</td>
<td>20-300mW</td>
<td>&lt;0.5W</td>
<td>2W-10W</td>
<td>5W-12W</td>
</tr>
<tr>
<td>External Memory</td>
<td>16b SDRAM</td>
<td>16b DDR2/DDR3</td>
<td>64b (LP)DDR3 w/ECC</td>
<td>64b (LP)DDR3 w/ECC</td>
</tr>
<tr>
<td>Operating system</td>
<td>CSL</td>
<td>TI RTOS</td>
<td>TI RTOS</td>
<td>Linux™/TI RTOS</td>
</tr>
</tbody>
</table>

J.Kasperek & P.J.Rajda © 2015 Katedra Elektroniki AGH
DSP on FPGA

- DSP Performance

- 7-Series
  - Virtex-7
  - Kintex-7

- 6-Series
  - Virtex-6
  - Spartan-6

- 4752 GMAC
- 2000 GMAC
- 770 GMAC
- 90 GMAC
- 33 GMAC

- Time

- Algorithm Complexity

- Multi-core DSP Architectures

* Peak performance for symmetric filters

J.Kasperek & P.J.Rajda © 2015 Katedra Elektroniki AGH
Example Application – 2x2 LTE Radio

Implemented using multiple devices

<table>
<thead>
<tr>
<th>#</th>
<th>Components</th>
<th># Devices</th>
<th>Power</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Multiple ASSPs, Protocol FPGA</td>
<td>7</td>
<td>~10.6W</td>
<td>~$316*</td>
</tr>
</tbody>
</table>

* 1KU resale (1HCY10)
Example Application – 2x2 LTE Radio

Single-chip Solution on Virtex-6

Virtex-6 LX75T

Integrates all DSP functionality and 6Gbps SERDES at lower cost, lower power, increased reliability and scales up to 2x4 and 4x4 within the same package

<table>
<thead>
<tr>
<th>#</th>
<th>Components</th>
<th># Devices</th>
<th>Power</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Multiple ASSPs, Protocol FPGA</td>
<td>7</td>
<td>~10.6W</td>
<td>~$316*</td>
</tr>
<tr>
<td>2</td>
<td>Virtex-6 LX75T</td>
<td>1</td>
<td>~5.8W</td>
<td>~$313*</td>
</tr>
</tbody>
</table>

* 1KU resale (1HCY10)
DSP implementation path on FPGA

https://www.youtube.com/user/eejournal

Chalk Talk
DSP implementation on FPGA

FPGAs are now HUGE!!
Faster Turnaround Times are Needed

Big DSP projects require expensive tools
• **Support for the following discrete-time filter structures:**
  - Finite impulse response (FIR) - Antisymmetric FIR - Transposed FIR - Symmetric FIR
  - Second-order section (SOS) infinite impulse response (IIR) Direct Form I - SOS IIR Direct Form I transposed - SOS IIR Direct Form II - SOS IIR Direct Form II transposed - Discrete-Time Scalar - Delay filter - Farrow (fractional delay) filter

• **Support for the following multirate filter structures:**
  - Cascaded Integrator Comb (CIC) interpolation - Cascaded Integrator Comb (CIC) decimation - Direct-Form Transposed FIR Polyphase Decimator - Direct-Form FIR Polyphase Interpolator - Direct-Form FIR Polyphase Decimator - FIR Hold Interpolator - FIR Linear Interpolator - Direct-Form FIR Polyphase Sample Rate Converter

• **Support for cascade filters (multirate and discrete-time)**
• **Generation of code that adheres to a clean HDL coding style**
• **Options for optimizing numeric results of generated HDL code**
• **Options for specifying parallel, serial (fully, partly or cascade), or distributed arithmetic architectures for FIR filter realizations**
• **Options for controlling the contents and style of the generated HDL code and test bench**
• **Test bench generation for validating the generated HDL filter code**
• **VHDL, Verilog, and ModelSim Tcl/Tk DO file test bench options**
• **Automatic generation of scripts for third-party simulation and synthesis tool**
Graphical user interface (GUI) accessible from Filter Design and Analysis
Many tutorials available directly from Mathworks
Key Features

• Generates synthesizable HDL code from Simulink models and **Embedded MATLAB™ code** for datapath implementations
• Generates synthesizable HDL code from Stateflow charts for Mealy and Moore finite-state machines and control logic implementations
• Generates VHDL code that is IEEE 1076 compliant and Verilog code that is IEEE 1364-2001 compliant
• Lets you create bit-true and cycle-accurate models that match your Simulink design specifications
• Lets you select from multiple HDL architectural implementations for commonly used blocks
• Lets you specify the subsystem for HDL code generation
• Enables you to reuse existing IP HDL code (with EDA Simulator Link products)
• Generates simulation and synthesis scripts
- RAM Blocks
- HDL Counter
- HDL FFT
- HDL Streaming FFT
- Bitwise Operators
Matlab HDL Coder™

MATLAB interfaces

Old Pain
Disjointed development flow
- Manual
- Difficult to share knowledge
- Slow to debug
- Chronic extra effort required

New Solution
Integrated automated flow
- Leverages existing MATLAB
- Enables cross-team effort
- Interactive debug
- “Plugs into” existing flow

EDA Simulator Link Product
Cosimulation Links

Algorithm
MATLAB
HDL
Implementation
Under Test
HDL Simulator
MATLAB Test Bench
MATLAB describes “Golden Reference”

EDA Digital Simulator
HDL

Stimulus
Data source → Algorithm Design → Analysis

MATLAB

Simulation
EDA

EDA Simulator Link product
EDA digital simulator

HDL

Algorithms
Scopes
Sources
M-code

HDL

Algorithms
Scopes
Sources
M-code

J.Kasperek & P.J.Rajda © 2015 Katedra Elektroniki AGH
## MATLAB interfaces

### Third-Party Products & Services

#### Products

<table>
<thead>
<tr>
<th>Products</th>
<th>Active-HDL</th>
<th>Riviera</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Comprehensive, integrated environment for digital IC design and verification</td>
<td>High-performance ASIC and large FPGA verification solution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Products</th>
<th>DSF Builder</th>
<th>Cadence Virtuoso AMS Designer Simulator</th>
<th>Cadence Virtuoso NeoCircuit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Quartz II and MATLAB/Simulink interface</td>
<td>Cosimulation of mixed-signal systems with MATLAB and Simulink</td>
<td>Analog and RF circuit sizing and optimization software</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Products</th>
<th>Synopsys Inc.</th>
<th>Synplicity Inc.</th>
<th>Xilinx, Inc.</th>
<th>Xilinx, Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Synopsys Inc.</td>
<td>Synplicity Inc.</td>
<td>Xilinx, Inc.</td>
<td>Xilinx, Inc.</td>
</tr>
<tr>
<td></td>
<td>Saber® Design and analysis of mixed-technology and mixed-signal systems</td>
<td>Synplify Pro® FPGA synthesis solution</td>
<td>AccelDSP High-level synthesis for DSP design</td>
<td>Xilinx System Generator™ for DSP Simulink blockset for bit- and cycle-accurate simulation and code generation for Xilinx FPGAs</td>
</tr>
</tbody>
</table>

**J.Kasperek & P.J.Rajda © 2015 Katedra Elektroniki AGH**
Many options available  
(budget, experience, time schedule, levels: System architect, DSP engineer, hardware/FPGA engineer)
Xilinx – System Generator for DSP (~5k$)

1. Describe the algorithm in mathematical terms,
2. Realize the algorithm in the design environment, initially using double precision,
3. Trim double precision arithmetic down to fixed point,
4. Translate the design into efficient hardware.
System Generator for DSP

- Library-based, visual data flow
- Polymorphic operators
- Arbitrary precision fixed-point
- Bit and cycle true modeling
- Multi-rate signal processing

- Seamlessly integrated with Simulink and MATLAB
  - Type and rate propagation
  - Test bench and data analysis

- Automatic code generation
  - Synthesizable VHDL
  - IP cores
  - HDL test bench
  - Project and constraint files
Simulink Extensions for HW

- HDL Configuration Wizard
- MATLAB compilation block
- HDL co-simulation
- Hardware in the loop co-simulation
DSP on FPGA - Altera

DSP Builder Design Flow

MATLAB/Simulink Domain
(System Simulation and Verification)

HDL/Hardware Domain
(Hardware Implementation/RTL Simulation)
## Floating-Point Comparison Table

<table>
<thead>
<tr>
<th>Feature</th>
<th>CPU</th>
<th>DSP</th>
<th>GPU</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>1X</td>
<td>10X</td>
<td>50X – 100X</td>
<td>100X</td>
</tr>
<tr>
<td>Power</td>
<td>1X</td>
<td>1X</td>
<td>5X – 10X</td>
<td>1X – 2X</td>
</tr>
<tr>
<td>I/Os</td>
<td>Very limited</td>
<td>Limited</td>
<td>Limited</td>
<td>High bandwidth</td>
</tr>
<tr>
<td>Longevity</td>
<td>Medium</td>
<td>Medium</td>
<td>Worst</td>
<td>Best</td>
</tr>
<tr>
<td>Complete system</td>
<td>Yes</td>
<td>No</td>
<td>No (requires CPU)</td>
<td>Yes</td>
</tr>
<tr>
<td>Tool flow</td>
<td>Software</td>
<td>Software</td>
<td>Software with multicore</td>
<td>Model-based</td>
</tr>
</tbody>
</table>

**FPGAs Offer Performance Advantage over CPUs/DSPs**

**FPGAs Offer Power Advantage over GPUs**
This is Calypto’s sixth annual High Level Synthesis survey report. 750 SoC, IC, and FPGA design professionals responded to the survey. The emphasis this year was on verification.
Types of Hardware Being Designed

- Wireless 3G/4G/LTE: 26%
- Video: 25%
- Imaging (ISP...): 22%
- Graphics: 21%
- Switch/Router: 20%
- Encryption: 18%
- Error Correction: 16%
- Wired LAN: 16%
- Not Designing Hardware: 7%
- Other: 25%

Calypso 2014
Synphony HLS provides a more automated implementation path from high-level models built from MATLAB descriptions and/or pre-built fixed-point IP.

Licence fee
~150k$/year
Synphony HLS features a library of synthesizable high-level IP functions targeted at multimedia and communications applications which is available in the Simulink model-based design and simulation environment. Using the Synphony MATLAB-synthesis block, designers can mix and match Math-language functions with Synphony high level IP for an easier and clearer specification of various system and algorithm behavior.
The difficulty and time consuming effort of creating models for system validation and functional verification is a major challenge in today’s system modeling and verification environments. Synphony HLS addresses this challenge by automatically creating C models for use in C-based verification.
AHDL ↔ MATLAB interface

MATLAB Workspace

VHDL Testbench

UUT Instantiation

Regular VHDL code
MATLAB Interface function calls

Arrays temporary storage

eval_string()
put_variable()
get_variable()
• Comprehensive model verification, data analysis, and results visualization
• Ability to compare theoretical algorithm with hardware implementation
• Effective use of mathematical formulas that may be difficult to implement with pure HDL
• Simple and efficient way of integration of HDL hardware models with the high-level model of the system in MATLAB
• Capability of generating sophisticated test vectors
• Execution of numerical computations that are not performed by designed hardware
• Extending the testbench with parts developed in MATLAB for faster algorithm execution and/or data visualization,
• An effective bidirectional data transfer between HDL simulator and MATLAB environment (millions of samples can be passed at ease).
• Ability to request any service from MATLAB directly from the HDL code, including sophisticated calculations and data visualization performed by the MATLAB graphical tools.
• **Software Requirements**
  MATLAB R14 or higher

• **Language Support**
  VHDL IEEE Std. 1076-1993
  Verilog IEEE Std. 1364-1995

• **Supported VHDL Data Types**
  STD_ULOGIC, STD_ULOGIC_VECTOR (up to 512 bits)
  STD_LOGIC and STD_LOGIC_VECTOR (up to 512 bits)
  BIT and BIT_VECTOR (up to 512 bits)
  SIGNED and UNSIGNED
  INTEGER
  REAL

• **Supported Verilog Types**
  Logic
  Integer
  Real
Every project with MATLAB should declare:

```
library aldec;
use aldec.matlab.all;
```
Description
The eval_string routine allows you to pass commands from the HDL Simulator (Active-HDL) to the MATLAB environment. MATLAB output is printed to the Console window (not to the MATLAB Command Window).

NOTE: Placing a semicolon after a MATLAB command disables the echo. This helps to limit excessive console output that could impair simulation performance.

Syntax
VHDL:
  eval_string("ml_cmd");
Verilog:
  $eval_string("ml_cmd");

ml_cmd
The command or expression to be sent to the MATLAB environment.
It is equivalent to typing a string in the MATLAB Console or the MATLAB Command Window.

Supported argument types (VHDL): string
Supported argument types (Verilog): string (i.e. a sequence of characters enclosed by double quotes). Note that string variables are not supported.
Description

The `put_variable` routine passes a variable from the HDL simulator (Active-HDL) to the MATLAB environment. The routine can be used only for scalar values and vectors (i.e. one-dimensional arrays).

Syntax
For an HDL variable (`hdl_var`) expressed as a scalar, a vector treated as an integer, an integer or a floating point number:

VHDL:
```vhdl
put_variable("ml_var_name", hdl_var);
```

For an HDL variable (`hdl_var`) expressed as a vector treated as fixed-point number:
VHDL:
```vhdl
put_variable("ml_var_name", hdl_var, point);
```
The name of the variable in the MATLAB environment. If the variable already exists, its value will be overwritten. If variable does not exist, it will be created and assigned.

Supported argument types (VHDL, Verilog): string

The HDL variable to be transferred to the MATLAB environment. The variable is read-only. The put_variable routine call will not change its value.

Supported argument types (VHDL): std_logic, std_logic_vector, signed, unsigned, bit, bit_vector, integer, real

Supported argument types (Verilog): reg (scalar or vector) net (scalar or vector), integer, real

The location of the binary point, starting from the least significant position. If set to 0, the number shall be treated as an integer value.
package Matlab is
  type TDims is array(POSITIVE range <> ) of integer;

-- procedure declaration

procedure put_variable(var_name : in string; var : in std_logic);
attribute foreign of put_variable: procedure is
  "VHPI $ALDEC/BIN/aldec_matlab_cosim.dll; put_variable_s";
...

-- procedure body

procedure put_variable(var_name : in string; var : in std_logic) is
begin
  end procedure;

-- VHPI – VHDL Programming Language Interface
Description
The get_variable routine allows you to pass a variable from the MATLAB environment to the HDL simulator (Active-HDL). The routine operates only for real scalar values (numbers of type double).

Syntax
For an HDL variable (hdl_var) expressed as a scalar, a vector treated as an integer, an integer or a floating point number:
VHDL:
```
get_variable("ml_var_name", hdl_var);
```
Verilog:
```
$gget_variable("ml_var_name", hdl_var);
```
For an HDL variable (hdl_var) expressed as a vector treated as fixed-point number:
VHDL:
```
get_variable("ml_var_name", hdl_var, point);
```
Verilog:
```
$gget_variable("ml_var_name", hdl_var, point);
```
get_variable("ml_var_name", hdl_var);
get_variable("ml_var_name", hdl_var, point);

ml_var_name
The name of a variable in the MATLAB environment. The MATLAB variable must be a real value of type double. Otherwise, the routine will fail. If no variable by the specified name is defined in MATLAB, an error message will be printed to the Console window. Supported argument types (VHDL, Verilog): string

hdl_var
The name of the HDL variable where the value should be stored. Supported argument types (VHDL): std_logic, std_logic_vector, signed, unsigned, bit, bit_vector, integer, real Supported argument types (Verilog): reg (scalar or vector), integer, real

point
The location of the binary point, starting from the least significant position. If set to 0, the number shall be treated as an integer value.
**Description**
Creates an array with the specified dimensions and returns the array identifier (array_id). If the array creation fails, 0 is returned. The number of array dimensions is currently limited to 6. It is recommended to remove the array from the memory with the `destroy_array` routine if it is no longer needed for calculations.

**Syntax**
VHDL: array_id = `create_array` ("name", ndims, dim_constr);
Verilog: array_id = `$create_array` ("name", dim_0, ... , dim_5);

*name*
*The name of the variable that will be used in the MATLAB environment.*

Supported argument types (VHDL, Verilog): **string**
(Note that string variables are not supported)

*ndims*
*The number of dimensions. Supported argument types (VHDL): integer*

*dim_constr*
*An array of integers specifying the number of elements for each dimension.*
Description
Removes an array from the memory

Syntax
VHDL:
   destroy_array (array_id);
Verilog:
   $destroy_array (array_id);

array_id
The array identifier.
**Description get_item**

Reads a floating-point number from the array and stores it in an HDL variable.

**Description put_item**

Changes an HDL value to a floating-point number and stores it in the specified element of the array.

**Syntax**

VHDL:
```
get_item (var, point, array_id, dim_sel);
put_item (var, point, array_id, dim_sel);
```

Verilog:
```
$get_item (var, point, array_id, sel_0, ..., sel_5);
$put_item (var, point, array_id, sel_0, ..., sel_5);
```

**var** - The name of an HDL variable containing the value to be stored in the selected array element.

**point** - The location of the binary point in the number (or the number of fractional positions).

**array_id** - Array identifier.

**dim_sel** - An array of integers specifying the location of the element in the array. Array indexing starts with 1.
**Description** ml2hdl

Reads the value of the variable specified with the `var_name` argument from the MATLAB environment. Returns the identifier of the array, where the variable is stored.

**Description** hdl2ml

Transfers an array specified by the array identifier to the MATLAB environment and stores it under the name previously specified with the `create_array` or `ml2hdl` routine.

**Syntax**

**VHDL:**

```vhdl
array_id := ml2hdl("var_name");
hdl2ml(array_id);
```

**Verilog:**

```verilog
array_id = $ml2hdl("var_name");
$hd12ml(array_id);
```

**var_name** - The name of a variable in the MATLAB environment. Supported argument types (VHDL, Verilog): **integer**

**array_id** - The table identifier. Supported argument types (VHDL, Verilog): **integer**


Description  put_simtime

Transfers simulation time to a MATLAB variable.

Syntax
VHDL:  
  put_simtime (var_name);
  put_simtime (var_name_t, var_name_u);
  put_simtime (var_name_t, var_name_u, class);
Verilog:  
  $put_simtime (var_name);
  $put_simtime (var_name_t, var_name_u);
  $put_simtime (var_name_t, var_name_u, class);

var_name
The name of a MATLAB variable where the current simulation time will be stored. The
time value will be expressed in seconds.
Supported argument types (VHDL): string (Verilog): string literal

var_name_t
The name of a MATLAB variable where the current simulation time will be stored. The
time value will be expressed in simulation time units.
Supported argument types (VHDL): string (Verilog): string literal
**AHDL ↔ MATLAB interface**

**passing the simulation time from AHDL to MATLAB**

**Example**

 Somewhere in user TB code VHDL:

```vhdl
put_simtime("HDL_sim_time", "ps");
eval_string(matlab_data_script);
get_variable("output_sample", data_sample);
```

**Content matlab_data_script**

```matlab
% send back to AHDL output sample - time is send in picoseconds
% values -1.0....1.0 are converted to U2 signed 14 bits
amplitude_scaling_factor = 2^13;
simulation_time_unit = 1e-12;
% get sample
how_many_samples_V = size(V_rx,2);
how_many_time_units_V = param_rx_duration/simulation_time_unit;
sample_index_V = round(HDL_sim_time*(how_many_samples_V/how_many_time_units_V));
if (sample_index_V < how_many_samples_V && sample_index_V > 0)
    output_sample = amplitude_scaling_factor * V_rx(sample_index_V);
else
    output_sample = 0;
    disp('ADC_V data index out of range');
end
```

J.Kasperek & P.J.Rajda © 2015 Katedra Elektroniki AGH
AHDL ↔ SIMULINK interface

- Direct link between Active-HDL and Simulink for bidirectional co-simulation and visualization through the Active-HDL Co-Sim block
- Instantiating VHDL entities, Verilog modules, or EDIF cells directly on a Simulink diagram by using HDL Black-Box blocks
- Generation of M-files describing HDL models
- Customization of HDL black-box parameters on a Simulink diagram
- Support of VHDL, Verilog, and EDIF netlists
- Support of VHDL generics and Verilog parameters
- Support of multiple HDL modules/entities or EDIF cells within one Simulink diagram
- Support of multiple synchronization signals (CLK/CE)
AHDL ↔ SIMULINK interface

- Possibility for defining different values for a sampling period in Simulink and a clock period in Active-HDL
- Conversion of the MATLAB floating- and fixed-point types to HDL types
- Unlimited position of binary point
- Interactive debug in Active-HDL during co-simulation
- Support of simulation parameters (simulator switches) in Active-HDL Co-Sim block
- Generating testbench during co-simulation for stand-alone simulation runs
- Logging simulation data in the format of the Accelerated and Standard Waveform Viewer for visualization of ports and internal signals of an HDL model
- Full support of Xilinx System Generator including both data types compatibility and ISE project generation
Example:
1. MATLAB `fdatool` filter design
2. Export filter coefficients from MATLAB
3. In AHDL start **IP Core Generator**
   -> configure filter template
4. Upload coefficients from MATLAB
5. AHDL generates synthesizable vhdl file
6. AHDL generates Testbench
Filter project example AHDL <=> MATLAB

Reading data process

Main process
TB verification

Writing data process

get_samples: process

variable array_id: integer; -- table id
variable ndims: integer; -- table dimensions
variable elem: integer; -- vector lengh
variable sample: std_logic_vector(15 downto 0);
variable dim_constr : TDims (1 to 2) := (1,1);
variable memory : BUFOR;

begin

eval_string("read_sound");
array_id := ml2hdl("ARRAY");
ndims := get_num_dims(array_id); -- get dimensions [1:1]
elem := get_dim(array_id, 2); -- get vector lengh
for I in 1 to elem loop
    get_item( sample, array_id, dim_constr );
    dim_constr(2):=dim_constr(2)+1;
    memory(I) := sample;
end loop;
--next part beside ->

data_size <= elem;
memory_in <= memory;
DataRead <= true;
destroy_array(array_id);
wait;
end process;
Plik read_sound.m

clear all;
fp = 44100;
t = (0:1/fp:0.02);
s1 = sin(2*pi*t*70);  % three signals
s2 = sin(2*pi*t*1000);
s3 = sin(2*pi*t*20000);
ARRAY=(32000/4)*(s1+s2+s3);  % ...added..
figure(1)
plot(t,ARRAY);
xlabel('Czas[s]');
ylabel('Amplituda');
title('Sygnał wejściowy');
DSP: process (CLK, DataRead) --
    variable J : integer := 1;
    begin
        if DataRead = true and rst = '0' then
            CE <= '1';
            if CLK = '1' and CLK'event then
                x <= memory_in (J); -- data write
                memory_out(J) <= y; -- data read
                if J < data_size then
                    J := J+1;
                else
                    DataWrite <= true;
                    -- write flag
                end if;
            end if;
        else
            CE <= '0';
        end if;
    end process;
write_data: process(DataWrite )
    variable samples : BUFOR;
    variable out_array: integer;
    begin
        if DataWrite = true then
            for K in 1 to rozmiar loop
                samples(K) := memory_out(K);
            end loop;
            out_array := create_array("out_samples", 2, (1,rozmiar) );
            for I in 1 to data_size loop
                put_item(samples(I), 0, out_array, (1,I));
            end loop;
            hdl2ml( out_array );
            eval_string( "write_sound" );
        end if;
    end process;

    -- Write data process
File write_sound.m

```matlab
    t = (0:1/fp:0.02);
    outputs(1,:) = out_samples;
    figure(2);
    plot(t,outputs, 'red');
```
Thank you!