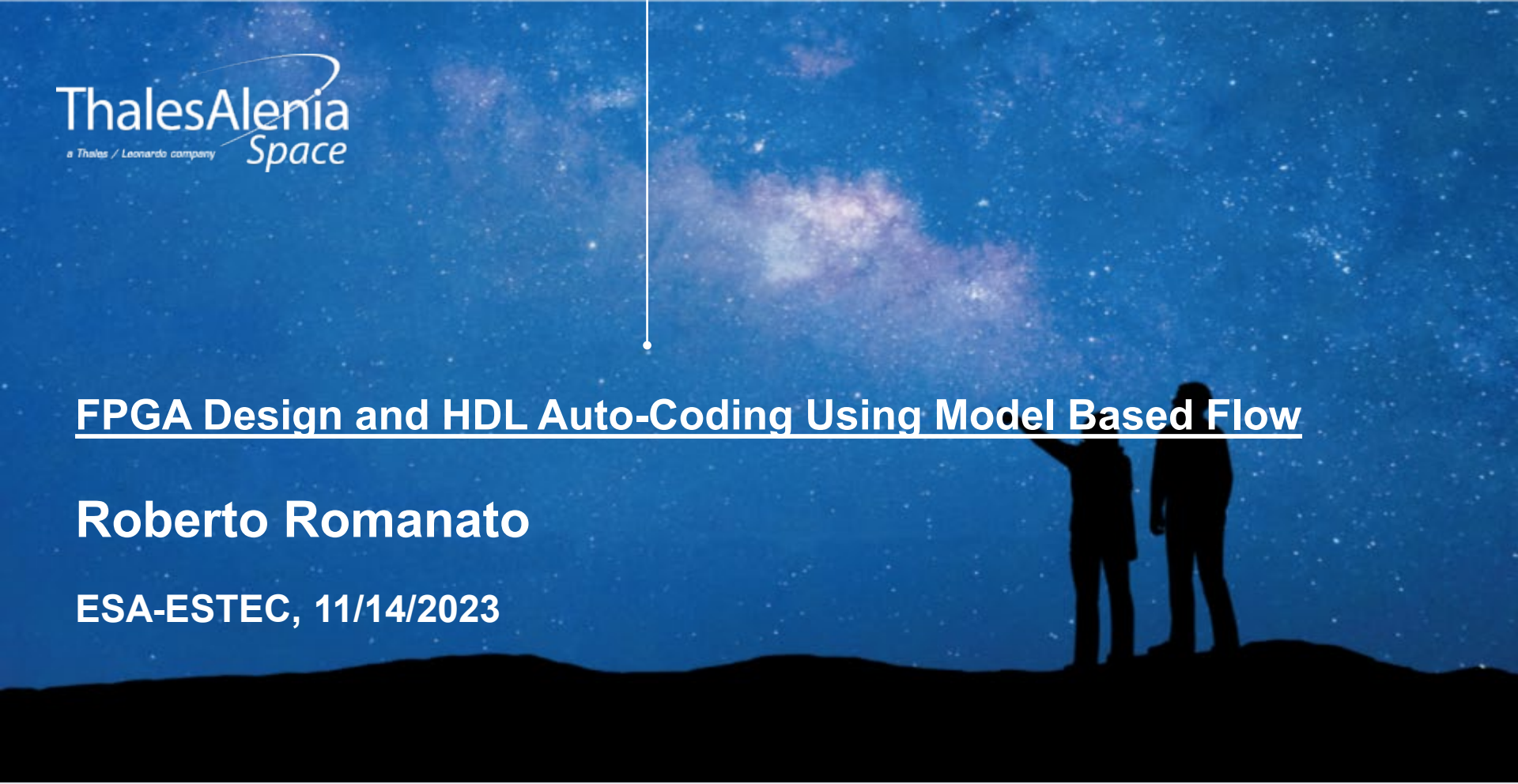


FPGA Design and HDL Auto-Coding Using Model Based Flow

Roberto Romanato

ESA-ESTEC, 11/14/2023

TABLE OF CONTENTS

Date: 11/15/2023
Ref: Not referenced
Template: 83230347-DOC-TAS-EN-007

INTRODUCTION

Date: 15/11/2023
Ref: Not referenced
Template: 83230347-DOC-TAS-EN-007

INTRODUCTION

///The continuous increase of FPGA complexity requires new tools and design flows

///Mathworks MATLAB/SIMULINK Design flow used since 2008

- Xilinx System Generator (2008-2012)
  - SIMULINK add-on with a kernel which permits to generate HDL Code
  - The SIMULINK model is designed using Xilinx System Generator library
  - Pros
    - Wide set of DSP building blocks (more or less the ones in the Xilinx catalogue)
    - A cascade of few components permits the implementation of quite complex systems
  - Cons
    - The generated code was not readable (more like a netlist than RTL)
    - The generation time was too long (hours for big designs)
    - If a small section of the design changed, the entire code had to be regenerated
    - The generated code is FPGA dependent
  - Conclusion: good tool for small designs with a cascade of DSP blocks, not versatile for complex FPGAs

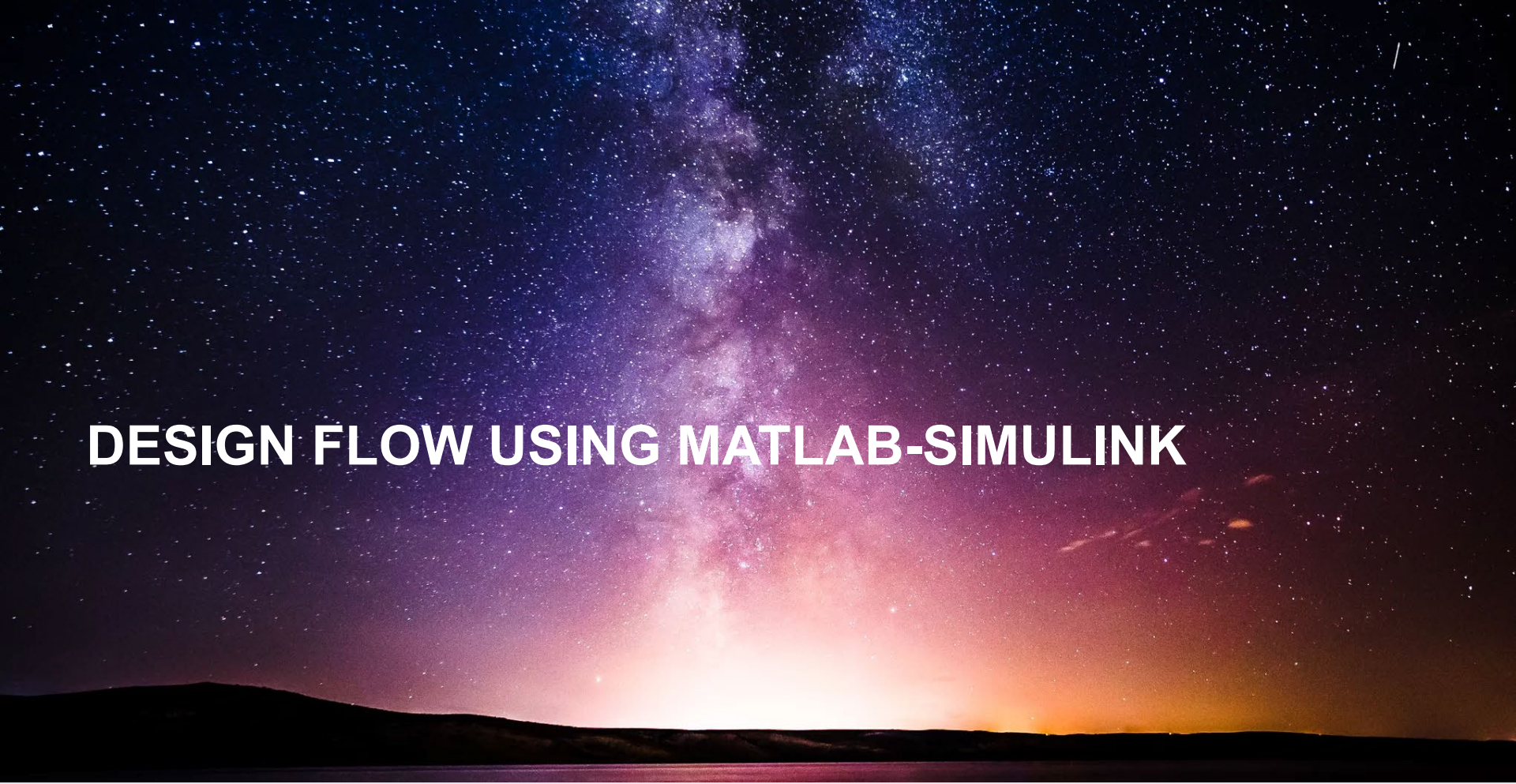
INTRODUCTION

///Mathworks MATLAB/SIMULINK Design flow used since 2008

- Mathworks HDL Coder (2013-Now)
  - SIMULINK toolbox
  - RTL code technology independent
  - The generation time is quite low (minutes for complex FPGAs)
  - If a section of the project changes, it is possible to regenerate the code only in the updated sub-system

///Reasons to use MATLAB/SIMULINK and HDL-Coder design flow

- Fully integrated platform from system to hardware level (i.e. hardware in the loop)
- There is a direct translation of the binary strings to decimal
- Speeds-up the debug since is easier to compare the FL-Point and FX-Point test vectors



DESIGN FLOW USING MATLAB-SIMULINK

Date: 15/11/2023
Ref: Not referenced
Template: 83230347-DOC-TAS-EN-007

DESIGN FLOW USING MATLAB-SIMULINK

/// MATLAB floating point simulator “FPGA oriented”

- Each sub-system that has to be implemented in VHDL is implemented as function
- Reflects architecture on FPGA (datapath and interfaces)

/// SIMULINK fixed point model (data and control path)

/// Radiation Aspects

- If the target FPGA requires TMR, the sensitive logic is grouped into few blocks (Control Paths)

/// The Test Bench is also implemented inside the SIMULINK model in order to have a unique model for testing the design

- The SIMULINK model of the TB can be translated in HDL code
- Synthesizable TB for HW debug/tests

/// Testing Scripts

/// HDL Code generation with SIMULINK “HDL Coder” toolbox

/// RTL simulation with QuestaSim and comparison w.r.t. Simulink model output

/// Synthesis and Place&Route

/// Test on hardware (Breadboard with FPGAs)

Date: 11/15/2023
Ref:
Template: 83230347-DOC-TAS-EN-007

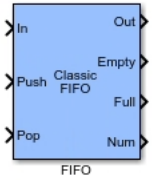
DESIGN FLOW USING MATLAB-SIMULINK

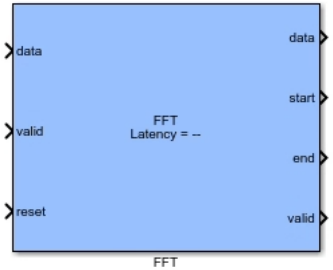
/// The SIMULINK model is RTL synthesis of our design

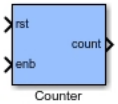
- FSMs, registers, counters, mathematical operators
- Quantization and configuration defined with MATLAB script

/// HDL-Coder supplies a SIMULINK library

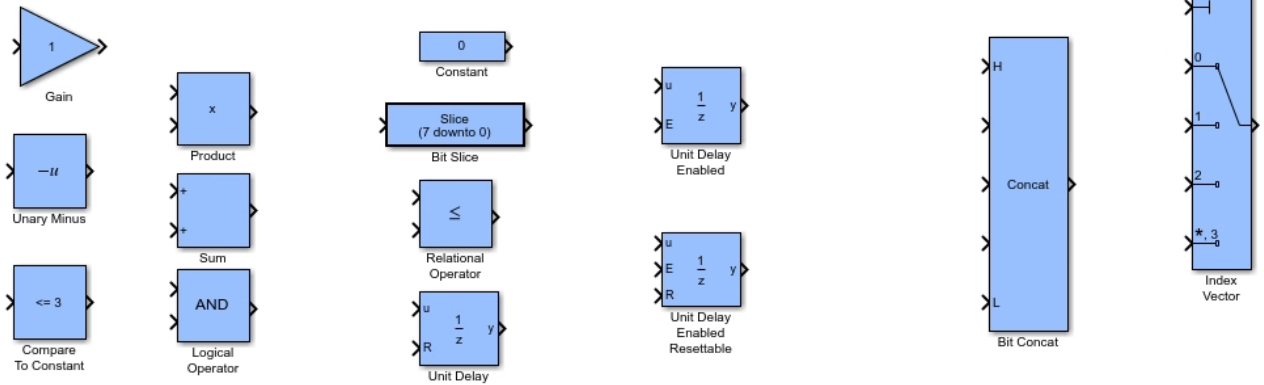
- Basic blocks (adders, multipliers, relational operators, etc.)
- DSP blocks (i.e. FFT)
- Memories (RAMs, ROMs, FIFOs)





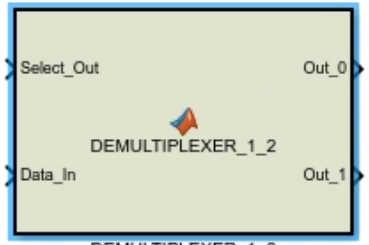


Note: In the SIMULINK model, the colored blocks are the ones that will be translated into HDL code



Date: 11/15/2023
Ref:
Template: 83230347-DOC-TAS-EN-007

DESIGN FLOW USING MATLAB-SIMULINK




```
function [Out_0,Out_1] = DEMULTIPLEXER_1_2(Select_Out,Data_In)

Out_0 = fi(0,0,1,0);
Out_1 = fi(0,0,1,0);

switch int8(Select_Out)
    case 0
        Out_0(1) = Data_In;
        Out_1(1) = 0;

    case 1
        Out_0(1) = 0;
        Out_1(1) = Data_In;

    otherwise
        Out_0(1) = 0;
        Out_1(1) = 0;
end
```

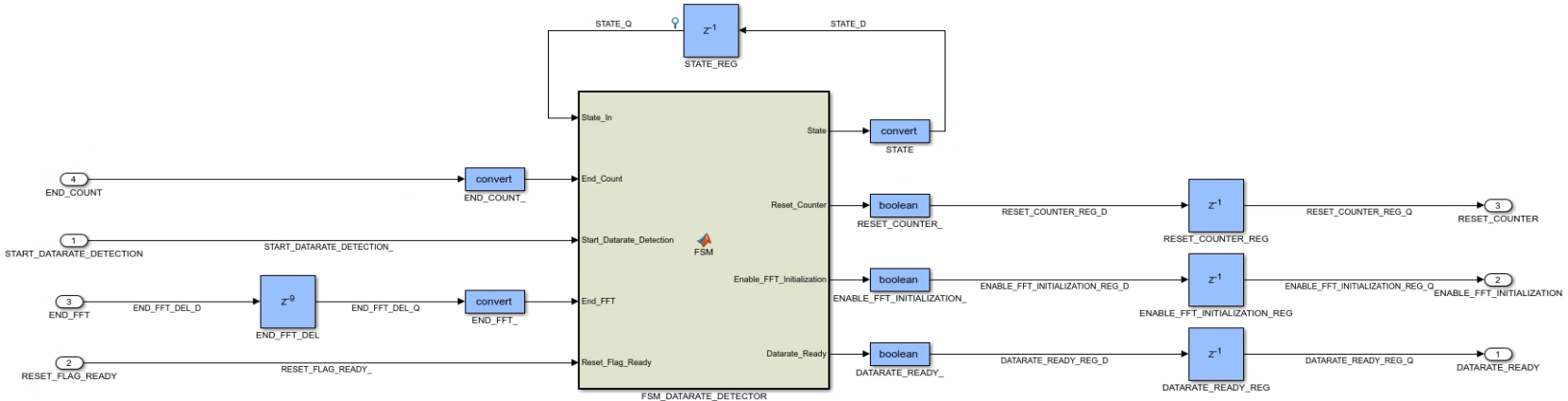
```
function [State,Reset_Counter,Enable_FFT_Initialization,Datarate_Ready] = FSM(State_In,End_Count,Start_Datarate_Detection,End_FFT,Reset_Flag_Ready)

State = fi(0,0,3,0);                                        % FSM state
Reset_Counter = fi(0,0,1,0);                                % Reset the counter of the FFT input samples
Datarate_Ready = fi(0,0,1,0);                               % Set to '1' the "FLAG_Reset_Flag_Ready" flag
Enable_FFT_Initialization = fi(0,0,1,0);                    % '1' for all the samples valid at the FFT input

switch uint8(State_In)
    case 0
        State(1) = 1;
        Reset_Counter(1) = 1;
        Datarate_Ready(1) = 0;
        Enable_FFT_Initialization(1) = 0;

    case 1
        if (Start_Datarate_Detection == 1)                  % Wait a new batch of samples to elaborate
            State(1) = 2;
            Reset_Counter(1) = 0;
            Enable_FFT_Initialization(1) = 1;
        else
            State(1) = 1;
            Reset_Counter(1) = 1;
            Enable_FFT_Initialization(1) = 0;
        end

        Datarate_Ready(1) = 0;

    case 2
        if (End_Count == 1)                                 % Read all the samples from the interface buffer
            State(1) = 3;
        else
            State(1) = 2;
        end

        Reset_Counter(1) = 0;
        Datarate_Ready(1) = 0;
        Enable_FFT_Initialization(1) = 1;
```


Date: 11/15/2023
Ref:
Template: 83230347-DOC-TAS-EN-007

DESIGN FLOW USING MATLAB-SIMULINK

///Finite State Machines

- Combinatorial Process (MATLAB Function)
- State (z^{-1})
- FSM outputs are sampled

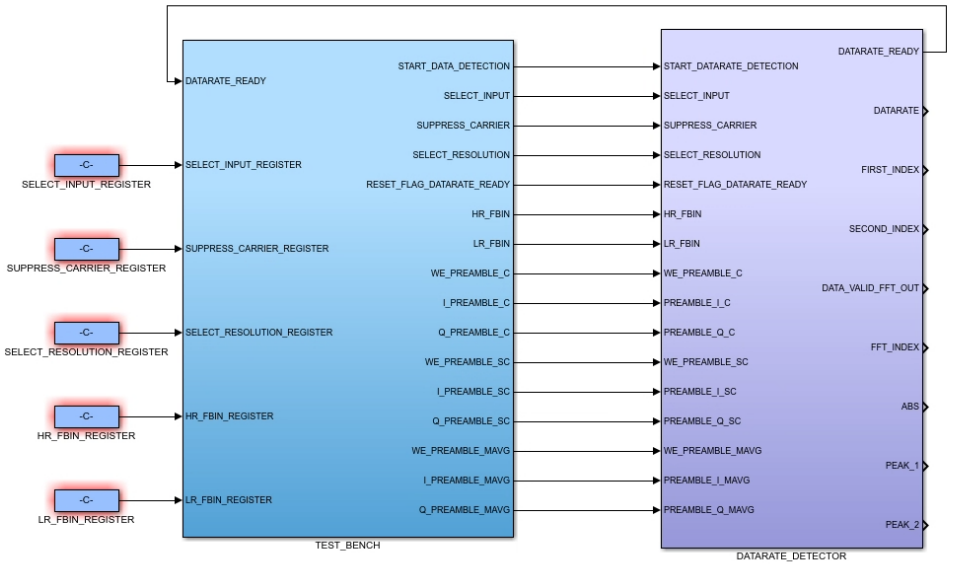


Date: 11/15/2023
Ref:
Template: 83230347-DOC-TAS-EN-007

CASE OF STUDY: DATARATE DETECTOR

Date: 15/11/2023
Ref: Not referenced
Template: 83230347-DOC-TAS-EN-007

DATARATE DETECTOR - OVERVIEW



///Datarate Detector

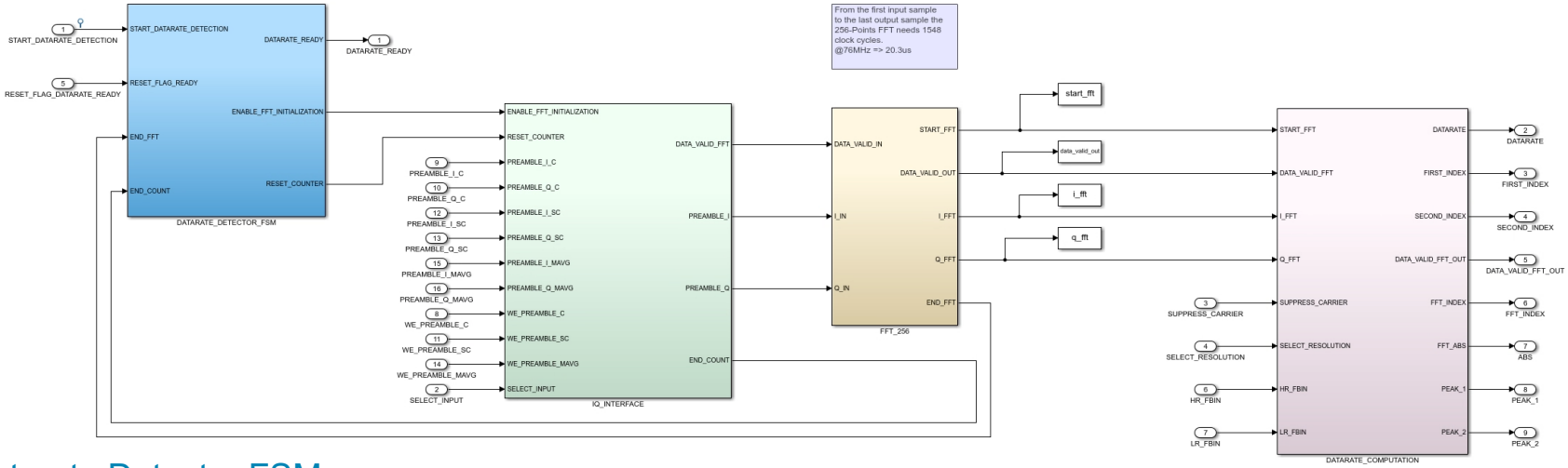
- DSP element of the Integrated Deep Space & Radio-Science Transponder (**ESA contract No. 4000125957/18/NL/FE**)
- System with heterogeneous elements
  - FSM
  - FFT
  - DSP elements
- Receives streams of samples and computes its datarate applying a dedicated algorithm

///Test Bench

- Supplies data to the Datarate Detector
- Implements the interaction with the Upper Layer (Leon2FT)

Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

DATARATE DETECTOR - ARCHITECTURE



/// Datarate Detector FSM

- Coordinates the operations
- Manages the UL interface

/// IQ Interface

- Manages the interface with other DSP blocks

/// 256-Points FFT (Burst)

/// Datarate Computation

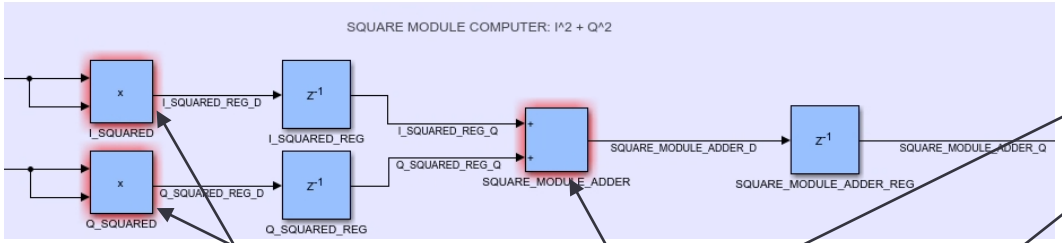
- Computes the final datarate using the FFT result

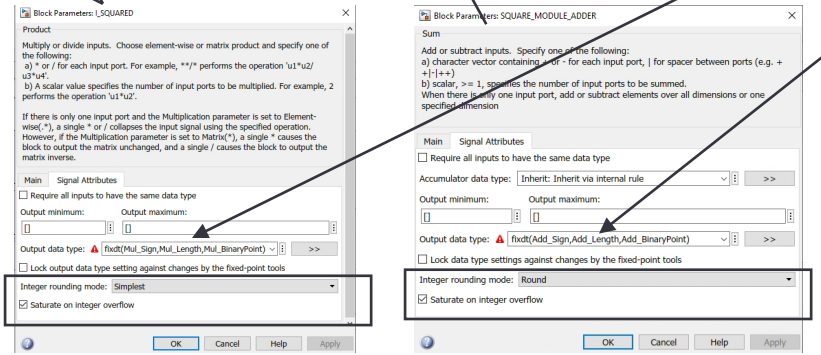
Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

DATARATE DETECTOR - CONFIGURATION AND QUANTIZATION

/// To be more efficient, a MATLAB script is used for :

- Quantizing the datapath
- Loading parameters/constants
- Parametric quantization





Quantization

```
% %%%%%%%%%%%%%%%%%%%% %
% Constants definition %
% %%%%%%%%%%%%%%%%%%%% %
N_FFT = 256;
HR_FBIN = 500;
LR_FBIN = 4000;
N_FFT_Length = log2(N_FFT);

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
% Datarate Computer Datapath Quantization %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
DataIn_Sign = 1;                                % Input samples
DataIn_Length = 10;
DataIn_BinPoint = 8;

Mul_Sign = 0;                                   % Square Module multiplier
Mul_Length = 27;
Mul_BinaryPoint = 8;

Add_Sign = 0;                                   % Square Module Adder
Add_Length = Mul_Length + 1;
Add_BinaryPoint = 8;

AddrSamp_Length = 8;                            % Samples index
AddrSamp_BinPoint = 0;

Datarate_Length = 19;                           % Datarate length
Resolution_Length = 12;                         % Frequency resolution length

Zero_Sign = 1;
Zero_Length = 18;
Zero_BinaryPoint = 8;
```

Rounding Method

```
set_param('DataRate_DetectorBurst/DATARATE_DETECTOR/DATARATE_COMPUTATION/SQUARE_MODULE_ADDER', 'RndMeth','Round');
set_param('DataRate_DetectorBurst/DATARATE_DETECTOR/DATARATE_COMPUTATION/SQUARE_MODULE_ADDER', 'SaturateOnIntegerOverflow','on');

set_param('DataRate_DetectorBurst/DATARATE_DETECTOR/DATARATE_COMPUTATION/I_SQUARED', 'RndMeth','Simplest');
set_param('DataRate_DetectorBurst/DATARATE_DETECTOR/DATARATE_COMPUTATION/Q_SQUARED', 'RndMeth','Simplest');
set_param('DataRate_DetectorBurst/DATARATE_DETECTOR/DATARATE_COMPUTATION/I_SQUARED', 'SaturateOnIntegerOverflow','on');
set_param('DataRate_DetectorBurst/DATARATE_DETECTOR/DATARATE_COMPUTATION/Q_SQUARED', 'SaturateOnIntegerOverflow','on');
```

DATARATE DETECTOR - SIMULATION

///MATLAB Script

- Run floating point simulator and save useful variables in ".mat" file
- Clear the workspace
- Run SIMULINK model
- Create comparison matrixes
- Analyse results

Note: the solver type is configured in «Fixed-step» with «Fixed-step size = 1»; in this way, each Simulink simulation point is a clock cycle

```
clc
clear all
close all

run DatarateDetector_Simulator.m

clear all

run ConfigureDatarate_Detector.m
run Configure_DatarateDetector_TB.m

Model = 'DataRate_DetectorBurst.slx';                                   % Set Simulink model name
SimOut = sim(Model);                                                    % Simulate simulink model in Fixed Step mode (i.e. each simulation point represents a clock cycle)

TEST = 4;

switch (TEST)
    case 0 % Validate Test Bench
        M = [state_tb state_dd rst start_det addr_we_pre we_pre data_valid_in enc_count addr_rd_samp valid i_pre q_pre];

    case 1 % Validate data rate detector
        M = [state_dd double(addr_we_pre) valid_in_fft i_in q_in data_valid_out i_fft q_fft abs_iq double(index_samp) data_ready en_th double(f_idx) double(s_idx) dr];

        Index = find(valid_in_fft == 1);

        M_SAMPLES = [addr_we_pre(Index) i_in(Index) q_in(Index)];

        Indexes = find(data_ready == 1);
        FFT_Indexes_FX = [f_idx(Indexes) s_idx(Indexes)];

        E = FFT_Indexes_FX - FFT_Indexes_FL;

    case 2
        M = [state_tb state_dd end_fft ena_flag rst_flag data_ready];

    case 3
        M = [state_tb rst_tb end_tb valid count_tb state_dd data_valid_out i_fft q_fft final_abs fft_valid_final double(index_samp) sel1 peak1 sel2 peak2 double(f_idx) double(s_idx) data_ready dr];
        Index = find(data_ready == 1);
        RRR = dr(Index(1:15:end));

    case 4
        Index = find(data_ready == 1);
        RRR = dr(Index(1:3:end));

    otherwise
        disp('No test is foreseen for this configuration')
end
```

HDL CODE GENERATION

Date: 15/11/2023
Ref: Not referenced
Template: 83230347-DOC-TAS-EN-007

HDL CODE GENERATION – OVERVIEW

/// Assign a name on every net
- Makes the generated code more readable

/// Code generation general settings
- Clock and Reset settings
- Ports and coding style

/// Hierarchy definition

/// Black Boxes

/// Test points for debug
- Preliminary code generation

Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

HDL CODE GENERATION – CLOCK AND RESET SETTINGS

PARAMETER	VALUE
Clock input port	clk
Clock edge	Rising/Falling
Reset input port	reset
Reset Asserted Level	Active-high/Active-low
Reset Type	Synchronous/Asynchronous

```
hdlset_param('DataRate_DetectorBurst','ClockInputPort','clk')
hdlset_param('DataRate_DetectorBurst','ClockEdge','Rising')
hdlset_param('DataRate_DetectorBurst','ResetInputPort','reset')
hdlset_param('DataRate_DetectorBurst','ResetAssertedLevel','Active-high')
```


Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

HDL CODE GENERATION – PORTS AND CODING STYLE

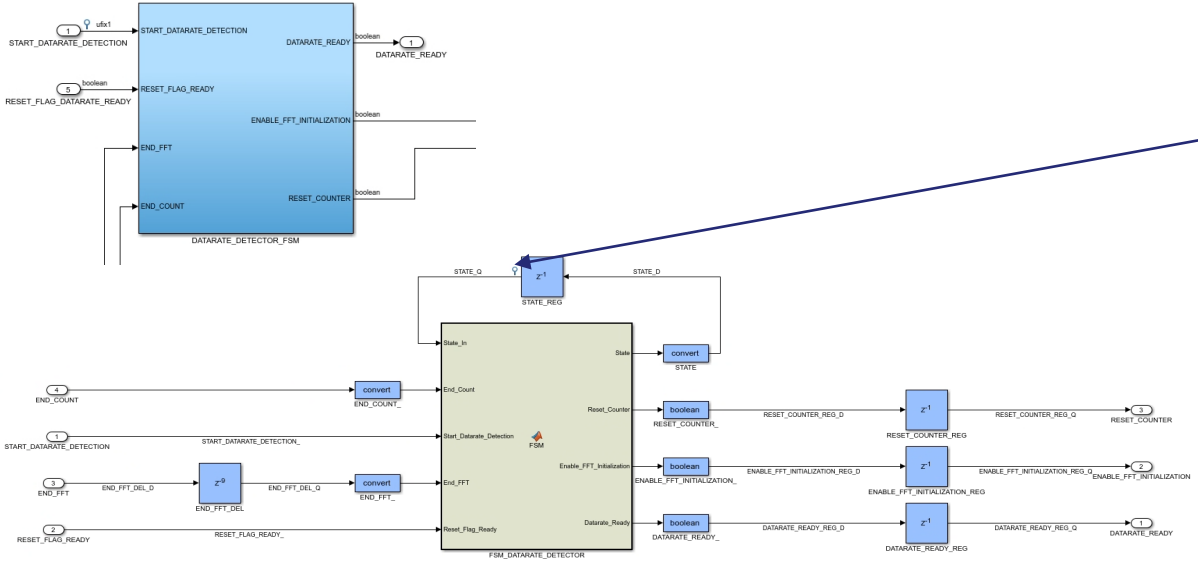
PARAMETER	VALUE
Minimize clock enables	On
Enable HDL DUT output port generation for test points	On
Minimize intermediate signals	On
Generate parameterized HDL code from masked subsystem	On
Use «rising_edge/falling_edge» style for registers	On
Code reuse	Atomic and Virtual

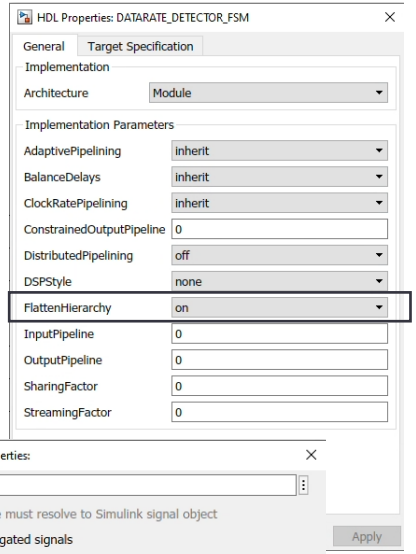
```
hdlset_param('DataRate_DetectorBurst','MinimizeClockEnables','On')
hdlset_param('DataRate_DetectorBurst','EnableTestpoints','On')
hdlset_param('DataRate_DetectorBurst','MinimizeIntermediateSignals','On')
hdlset_param('DataRate_DetectorBurst','MaskParameterAsGeneric','On')
hdlset_param('DataRate_DetectorBurst','UseRisingEdge','On')
hdlset_param('DataRate_DetectorBurst','SubsystemReuse','Atomic and Virtual')
```

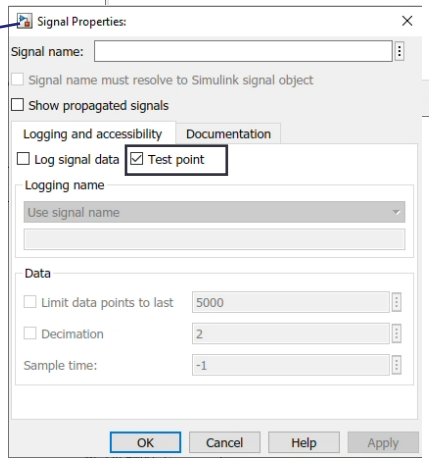
HDL CODE GENERATION – HIERARCHY/TEST POINTS

/// Define which blocks in the SIMULINK model have to be converted in a HDL file

- HDL Coder generates a VHDL file for each SIMULINK Sub-System and MATLAB-Function
- If a Sub-System or MATLAB-Function does not need to be converted in a HDL file, then the "Flatten" option must be enabled using the HDL Block Properties – in this way the sub-system is contained within the higher level of the hierarchy
- SIMULINK IP (i.e. FFT/IFFT) can't be "Flattened"







Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

HDL CODE GENERATION – EXAMPLE 1

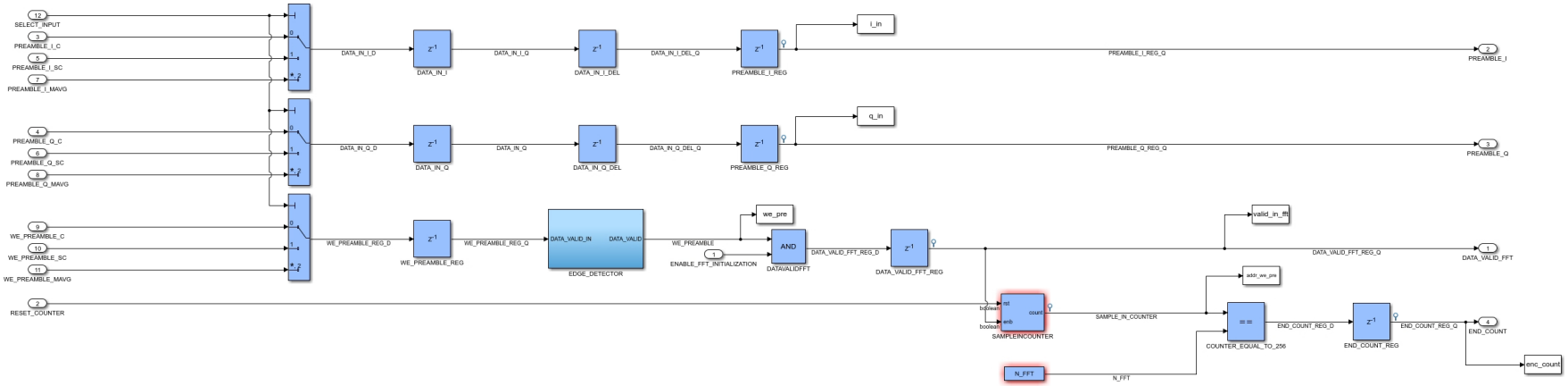
```
ENTITY DATARATE_DETECTOR_FSM IS
  PORT( clk                              :   IN    std_logic;
        reset                            :   IN    std_logic;
        START_DATARATE_DETECTION         :   IN    std_logic;  -- ufix1
        RESET_FLAG_READY                 :   IN    std_logic;
        END_FFT                          :   IN    std_logic;
        END_COUNT                        :   IN    std_logic;
        DATARATE_READY                   :   OUT   std_logic;
        ENABLE_FFT_INITIALIZATION        :   OUT   std_logic;
        RESET_COUNTER                    :   OUT   std_logic;
        tp_STATE_Q                       :   OUT   std_logic_vector(2 DOWNTO 0)  -- ufix3 Testpoint port
        );
END DATARATE_DETECTOR_FSM;
```

```
 FSM_DATARATE_DETECTOR_output : PROCESS (END_COUNT_out1, END_FFT_out1, RESET_FLAG_READY_1, START_DATARATE_DETECTION_1,
      STATE_Q)
 BEGIN
   -- FSM state
   -- Reset the counter of the FFT input samples
   -- Set to '1' the "FLAG_Reset_Flag_Ready" flag
   -- '1' for all the samples valid at the FFT input
   CASE STATE_Q IS
     WHEN "000" =>
       State <= to_unsigned(16#1#, 3);
       Reset_Counter_1 <= '1';
       Datarate_Ready_1 <= '0';
       Enable_FFT_Initialization_1 <= '0';
     WHEN "001" =>
       IF START_DATARATE_DETECTION_1 = '1' THEN
         -- Wait a new batch of samples to elaborate
         State <= to_unsigned(16#2#, 3);
         Reset_Counter_1 <= '0';
         Enable_FFT_Initialization_1 <= '1';
       ELSE
         State <= to_unsigned(16#1#, 3);
         Reset_Counter_1 <= '1';
         Enable_FFT_Initialization_1 <= '0';
       END IF;
       Datarate_Ready_1 <= '0';
     WHEN "010" =>
       IF END_COUNT_out1 = '1' THEN
         -- Read all the samples from the interface buffer
         State <= to_unsigned(16#3#, 3);
       ELSE
         State <= to_unsigned(16#2#, 3);
       END IF;
       Reset_Counter_1 <= '0';
       Datarate_Ready_1 <= '0';
       Enable_FFT_Initialization_1 <= '1';
     WHEN "011" =>
       -- Wait FFT ultimation (it is also considered the delay for searching the peaks)
       IF END_FFT_out1 = '1' THEN
         State <= to_unsigned(16#4#, 3);
         Datarate_Ready_1 <= '1';
       ELSE
```

```
STATE_REG_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    STATE_Q <= to_unsigned(16#0#, 3);
  ELSIF rising_edge(clk) THEN
    STATE_Q <= STATE_D;
  END IF;
END PROCESS STATE_REG_process;
```


Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

HDL CODE GENERATION



Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

HDL CODE GENERATION – EXAMPLE 2

```
ENTITY IQ_INTERFACE IS
  PORT( clk                          :   IN    std_logic;
        reset                        :   IN    std_logic;
        ENABLE_FFT_INITIALIZATION    :   IN    std_logic;
        RESET_COUNTER                :   IN    std_logic;
        PREAMBLE_I_C                 :   IN    std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        PREAMBLE_Q_C                 :   IN    std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        PREAMBLE_I_SC                :   IN    std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        PREAMBLE_Q_SC                :   IN    std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        PREAMBLE_I_MAVG              :   IN    std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        PREAMBLE_Q_MAVG              :   IN    std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        WE_PREAMBLE_C                :   IN    std_logic;
        WE_PREAMBLE_SC               :   IN    std_logic;
        WE_PREAMBLE_MAVG             :   IN    std_logic;
        SELECT_INPUT                 :   IN    std_logic_vector(1 DOWNTO 0);  -- ufix2
        DATA_VALID_FFT               :   OUT   std_logic;
        PREAMBLE_I                   :   OUT   std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        PREAMBLE_Q                   :   OUT   std_logic_vector(9 DOWNTO 0);  -- sfix10_En8
        END_COUNT                    :   OUT   std_logic;
        tp_DATA_VALID_FFT_REG_Q      :   OUT   std_logic;  --  Testpoint port
        tp_END_COUNT_REG_Q           :   OUT   std_logic;  --  Testpoint port
        tp_PREAMBLE_I_REG_Q          :   OUT   std_logic_vector(9 DOWNTO 0);  -- sfix10_En8 Testpoint port
        tp_PREAMBLE_Q_REG_Q          :   OUT   std_logic_vector(9 DOWNTO 0);  -- sfix10_En8 Testpoint port
        tp_SAMPLE_IN_COUNTER         :   OUT   std_logic_vector(8 DOWNTO 0)  -- ufix9 Testpoint port
        );
END IQ_INTERFACE;
```

```
DATA_IN_I_D <= PREAMBLE_I_C_signed WHEN SELECT_INPUT_unsigned = to_unsigned(16#0#, 2) ELSE
    PREAMBLE_I_SC_signed WHEN SELECT_INPUT_unsigned = to_unsigned(16#1#, 2) ELSE
    PREAMBLE_I_MAVG_signed;

DATA_IN_I_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    DATA_IN_I_Q <= to_signed(16#000#, 10);
  ELSIF rising_edge(clk) THEN
    DATA_IN_I_Q <= DATA_IN_I_D;
  END IF;
END PROCESS DATA_IN_I_process;
```

```
STATE_2_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    STATE_Q <= to_unsigned(16#0#, 2);
  ELSIF rising_edge(clk) THEN
    STATE_Q <= STATE_D;
  END IF;
END PROCESS STATE_2_process;


FSM_EDGE_DETECTOR_output : PROCESS (DATAVALID_IN_out1, STATE_Q)
BEGIN
  -- FSM state
  CASE STATE_Q IS
    WHEN "00" =>
      State <= to_unsigned(16#1#, 2);
      Data_Valid <= '0';
    WHEN "01" =>
      IF DATAVALID_IN_out1 = '1' THEN
        State <= to_unsigned(16#2#, 2);
        Data_Valid <= '1';
      ELSE
        State <= to_unsigned(16#1#, 2);
        Data_Valid <= '0';
      END IF;
    WHEN "10" =>
      IF DATAVALID_IN_out1 = '1' THEN
        State <= to_unsigned(16#2#, 2);
      ELSE
        State <= to_unsigned(16#1#, 2);
      END IF;
      Data_Valid <= '0';
    WHEN OTHERS =>
      State <= to_unsigned(16#0#, 2);
      Data_Valid <= '0';
  END CASE;
END PROCESS FSM_EDGE_DETECTOR_output;
```


Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

HDL CODE GENERATION – CHECK

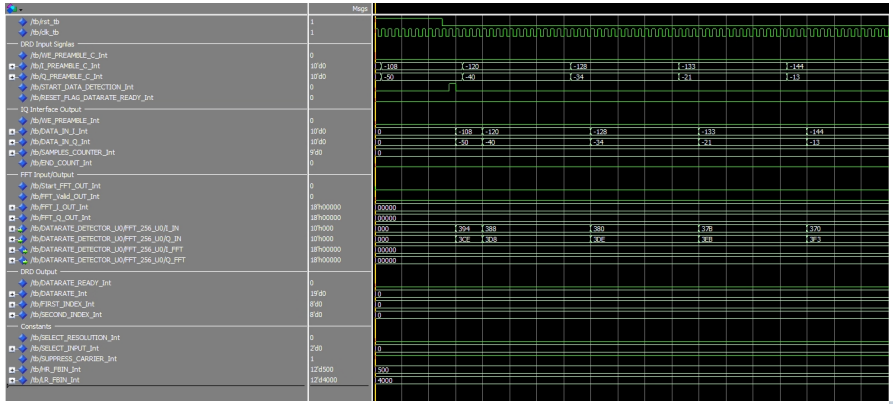
/// SIMULINK Test Bench is converted into HDL code

/// HDL Coder generates also a “.do” file to help the design compilation

/// Check HDL code output

- After a transient, the difference between both, SIMULINK and QuestaSim simulation must be ‘0’

/// Code Coverage



Name	Specified path	Full path	Type	Stmt Count	Stmt Hits	Stmt %	Stmt Graph	Branch Count	Branch Hits	Branch %	Branch Graph	Condition Count
sim	vsim.wlf	C:/Proge…										
MINRESRX2_BUTTERFLY.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	77	77	100.00		21	21	100.00		
IQ_INTERFACE.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	76	76	100.00		65	62	95.38		
MINRESRX2FFT_OUTMux.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	36	36	100.00		8	8	100.00		
Complex3Multiply.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	72	68	94.44		37	33	89.19		
std_logic_textio.vhd	C:/Software/q…	C:/Softw…	vhdl									
TEST_BENCH.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
FFT_256.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	12	12	100.00		2	2	100.00		
TEST_BENCH_pkg.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
mti_std_logic_unsigned.vhd	C:/Software/q…	C:/Softw…	vhdl									
FFT_256_pkg.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
mti_numeric_std.vhd	C:/Software/q…	C:/Softw…	vhdl									
DATARATE_DETECTOR.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	1	1	100.00						
FSM_TB.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
mti_std_logic_arith.vhd	C:/Software/q…	C:/Softw…	vhdl									
FFT.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	37	33	89.19		21	18	85.71		
DATARATE_COMPUTATION.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	118	118	100.00		113	109	96.46		
MINRESRX2FFT_BTFSEL.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	94	94	100.00		14	14	100.00		
I_SAMPLES.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
MINRESRX2FFT_MEMORY.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	154	154	100.00		50	50	100.00		
TB.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
standard.vhd	C:/Software/q…	C:/Softw…	vhdl									
MINRESRX2FFT_MEMSEL.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	56	56	100.00		13	13	100.00		
DATARATE_DETECTOR_FSM.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	59	57	96.61		35	34	97.14		
Q_SAMPES.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
MINRESRX2FFT_CTRL.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	366	345	94.26		65	57	87.69		
WE_PREAMBLE.vhd	C:/Progetti/ID…	C:/Proge…	vhdl									
textio.vhd	C:/Software/q…	C:/Softw…	vhdl									
stdlogic.vhd	C:/Software/q…	C:/Softw…	vhdl									
SimpleDualPortRAM_generic.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	6	6	100.00		4	4	100.00		
TWDLROM.vhd	C:/Progetti/ID…	C:/Proge…	vhdl	124	110	88.71		52	46	88.46		

PROS AND CONS

Date: 15/11/2023
Ref: Not referenced
Template: 83230347-DOC-TAS-EN-007

PROS AND CONS

///The designer's experience has a key role

///A good architecture results in better generated code and better performances (timing, resources)

///Pros

- Low learning curve (SIMULINK)
- Low development time
  - Graphical representation
  - The debug is faster
    - Automatic data representation conversion from binary to decimal
    - Test vectors generation, simulation and check are performed in the same environment
    - Quantization, rounding modes, hierarchy of an entire FPGA can be changed only updating a script

PROS AND CONS

///Cons

- DSP blocks like FFT and IFFT generate (too) many files
  - 1024-Points streaming FFT generates more than 100 files
- Even if is possible to reduce the number of generated signals, there are many “un-necessary” signals which are instantiated
- HDL Coder doesn’t permit to insert attributes (useful for TMR)
- The are still small bugs in code generation which have to be managed by the designer
  - For example “Data Type Conversion” which inserts spurious multiplexer

Date:
Ref:
Template: 83230347-DOC-TAS-EN-007

CONCLUSIONS

Date: 15/11/2023
Ref: Not referenced
Template: 83230347-DOC-TAS-EN-007

CONCLUSIONS

/// The presented design flow is used also for designing entire FPGAs (ex. OBP)

- The same code is mapped in different technologies for Bread Boarding activities
  - Xilinx Virtex 6
  - Xilinx RFSoC ZU28DR (ZCU111 Board)
  - Microsemi Polarfire MPF300
  - **Microsemi Polarfire RTPF500**

/// A single designer can design and manage several FPGAs

THANK YOU!

roberto.romanato@thalesaleniaspace.com