RAM Initialization and ROM Emulation in ProASIC$^{\underline{PLUS}}$ Devices

Table of Contents

Introduction

ProASIC$^{\underline{PLUS®}}$ field programmable gate arrays (FPGAs) are re-programmable and live at power-up and therefore offer a single-chip solution for programmable logic applications. The ProASIC$^{\underline{PLUS}}$ device architecture includes dedicated embedded SRAM memory blocks. The ProASIC$^{\underline{PLUS}}$ embedded SRAM blocks have the flexibility of being initialized through the JTAG port. The initialized memory blocks can also be used as ROM.

Libero integrated design environment (IDE) software allows users to define the initialized RAM data for simulation purposes. This document describes the procedure and requirements for initializing the memory blocks of the device using the test access port (JTAG TAP).

RAM Initialization

The initialization data needs to be loaded into the ProASIC$^{\underline{PLUS}}$ FPGA through JTAG pins. Interfacing the TAP to the device logic core is done through an embedded hard macro called UJTAG. The UJTAG macro is implemented in unused I/O tiles, and therefore, it does not consume any of the core logic tiles. The core-accessible ports of the UJTAG macro provide access to the contents of the TAP instruction register (IR) and TAP controller states as well as the JTAG pins TDI, TDO, and TCK. For more information on the UJTAG macro and its ports, refer to *AC202: ProASICPLUS PLL Dynamic Reconfiguration Using JTAG Application Note*.

Values 16 to 127 of the JTAG TAP IR are user-defined OPCODEs and are not reserved. The user can select any of these OPCODEs to define instructions for different stages of the RAM initialization process. For example, the designer can define an IR OPCODE of 34 to start the RAM initialization process and 35 to stop it. This is discussed further in the following sections.

UJTAG and RAM Block Interface

A user interface is required to receive the user command, initialization data, and clock from the UJTAG macro. The interface must synchronize and load the data into the correct RAM block of the design. The main outputs of the user interface block are the following:

- Memory block chip select: Selects a memory block of the design to be initialized. The chip select signals for each memory block can be generated from different user-defined opcodes or simple logic such as a ring counter (see Figure 1).
- Memory block write address: Identifies the address of the memory cell that needs to be initialized.
- Memory block write data: The interface block receives the data serially from the UTDI port of the UJTAG macro and loads it in parallel into the write data ports of the memory blocks.
- Memory block write clock: Drives the WCLK of the memory block and synchronizes the write data, write address, and chip-select signals.

Figure 1 shows the user interface between the UJTAG and the memory blocks.

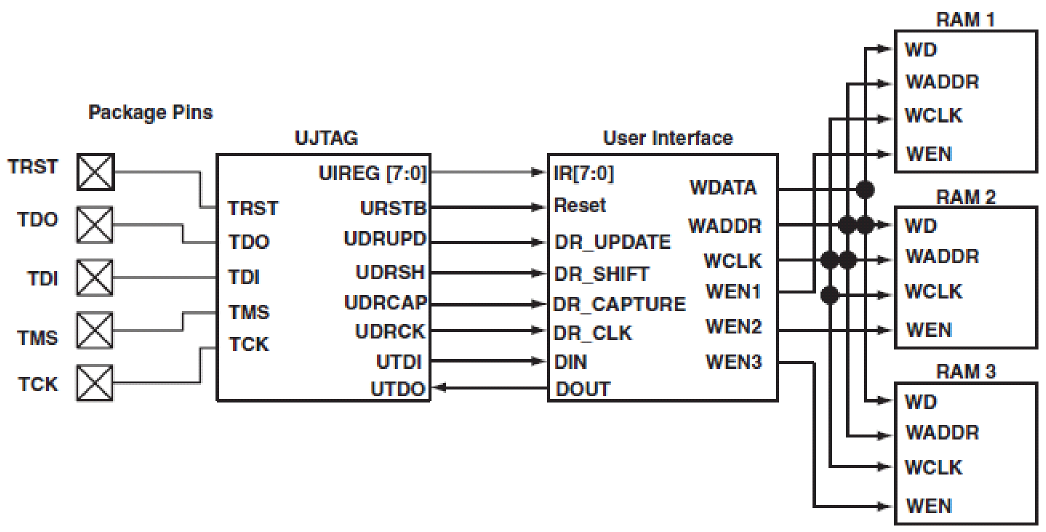


***Figure 1* • Interfacing TAP Ports and RAM Blocks**

An important component of the interface between the UJTAG macro and the RAM blocks is a serial-in/parallel-out shift register. The width of the shift register must equal to the data width of the RAM blocks. The data arrives serially through the UTDI output of the UJTAG macro. The data needs to be shifted into a shift register clocked by the JTAG clock (provided at the UDRCK output of the UJTAG macro). After the shift register is fully loaded, the data needs to be transferred to the write data port of the RAM block. In order to synchronize the write data loading with the write address and write clock, the output of the shift register can be pipelined before driving the RAM block.

The write address can be generated in different ways. It can be imported through the TAP using a different instruction opcode and another shift register or generated internally using a simple counter. Using a counter to generate the address bits and sweep through the address range of the RAM blocks is recommended, since it reduces the complexity of the user interface block and the board-level JTAG driver. Moreover, using an internal counter for address generation speeds up the initialization procedure since the user only needs to import the data through the JTAG ports.

The designer may use different methods to select among the multiple RAM blocks. Using counters along with De-Multiplexers is one approach to set the Write Enable signals. Basically, the number of RAM blocks needing initialization determines the most efficient approach. For example, if all the blocks are initialized with the same data, one enable signal is enough to activate the write procedure for all of them at the same time. Another alterative is to use different opcodes to initialize each memory block. For a small number of RAM blocks, using counters is an optimal choice. For example, a ring counter can be used to select among multiple RAM blocks. The clock driver of this counter needs to be controlled by the address generation process.

Once the addressing of one block is finished, a clock pulse is sent to the ring counter to select the next memory block.

Figure 2 shows a simple block diagram of an interface block between UJTAG and RAM blocks.

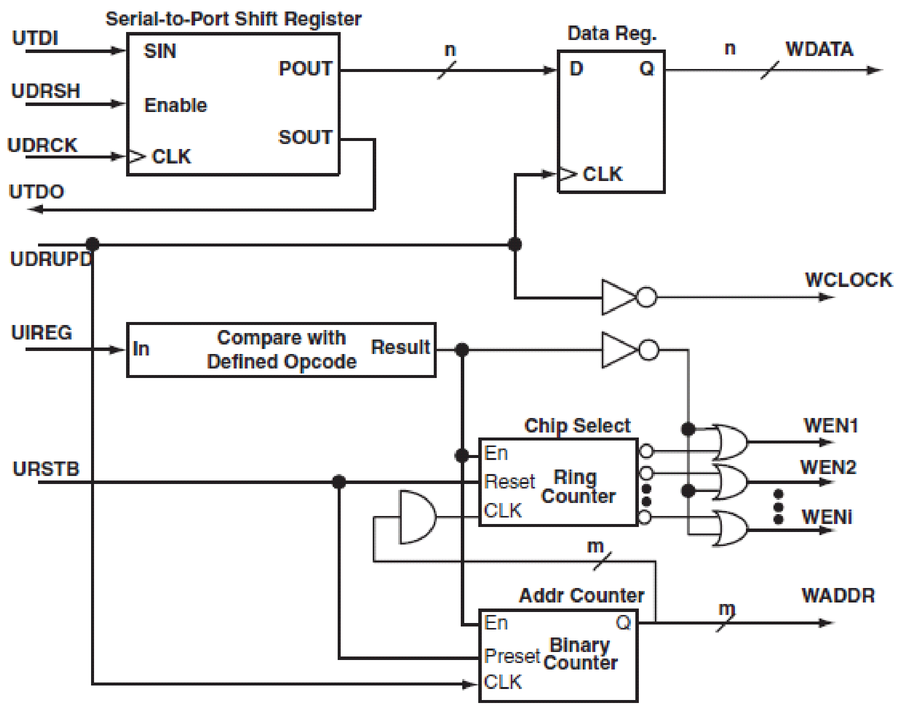


Figure 2 • **The Block Diagram of a Sample User Interface**

In Figure 2, the shift register is enabled by the UDRSH output of the UJTAG macro. The counters and chip-select outputs are controlled by the value of the TAP IR. The comparison block compares the UIREG value with the start initialization opcode value (defined by the user). If the result is true, the counters start to generate addresses and activate the WEN inputs of appropriate RAM blocks.

The UDRUPD output of the UJTAG macro, shown in Figure 2 is used for generating the write clock (WCLK) and synchronizing the data register and address counter with WCLK. UDRUPD is high, when the TAP controller is in the Data Register Update state, which is an indication of completing the loading of one data word. Once the TAP controller goes into the Data Register Update state, the UDRUPD output of the UJTAG macro goes high, and therefore, the pipeline register and the address counter place the proper data and address on the outputs of the interface block. Meanwhile, WCLK is defined as the inverted UDRUPD. This provides enough time (equal to the UDRUP high time) for the data and address to place at the proper ports of the RAM block before the rising edge of WCLK. The inverter is not required, if the RAM blocks are clocked at the falling edge of the write clock, as shown in Figure 3 on page 4.

Designers can use placement constraints to group the interface block cells in a predefined area of the die. This will isolate the RAM initialization logic from the main design. For information on ProASIC^{PLUS} placement constraints, refer to the *Designer v9.1 user guide*.

An Example of RAM Initialization

This section presents a sample design in which a 4 × 4 RAM block is being initialized through JTAG ports. A test feature is implemented in the design to read back the contents of the RAM after initialization to verify the procedure.

The interface block of this example performs two major functions – initialization of the RAM block and running a test procedure to read back the contents. The clock output of the interface is either the write clock (for initialization) or the read clock (for reading back the contents). The Verilog code for the interface block is included in the "Appendix: Interface Block" on page 6.

In this example, declaring the JTAG port in the top-level module is for simulation purposes. Without those ports, instantiation of the UJTAG macro is enough for designer software to connect the TAP to the inputs of the UJTAG during place-and-route. Declaration of the JTAG ports in the top module of the design will cause the synthesis tool to instantiate input and output buffers for these ports, which will cause error in designer, since these are not regular I/O pins. To avoid such problems, the user can remove the TAP from the top module during synthesis or remove the I/O buffers from the net list before importing them into designer. However, the first solution is easier and will not break the design flow integration.

Figure 3 shows the simulation results for the initialization step of the example design.

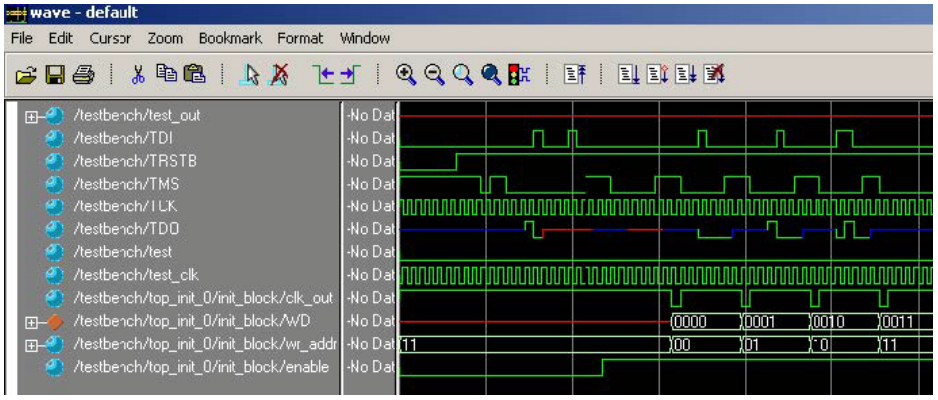
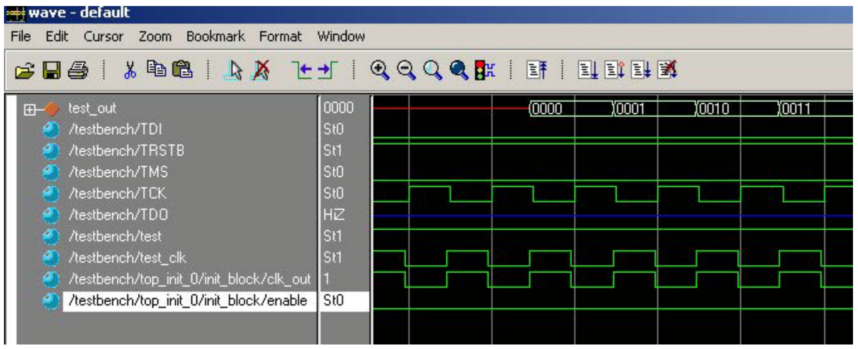

wave - default
File Edit Cursor Zoom Bookmark Format Window
/testbench/test_out
/testbench/TDI
/testbench/TRSTB
/testbench/TMS
/testbench/TCK
/testbench/TDO
/testbench/test
/testbench/test_clk
/testbench/top_init_0/init_block/clk_out
/testbench/top_init_0/init_block/WD
/testbench/top_init_0/init_block/wr_addr
/testbench/top_init_0/init_block/enable
-No Data-
0000
0001
0010
0011
11
00
01
10
11

Figure 3 • Simulation of Initialization Step

The CLK_OUT signal, which is the clock output of the interface block, is the inverted DR_UPDATE output of the UJTAG macro. It is clear that it gives sufficient time (the TAP controller is in the Data Register Update state) for the write address and data to become stable before loading them into the RAM block.

Figure 4 shows the test procedure of the example. The data read back from the memory block matches the written data, thus verifying the correct functionality of the design.



Note: The CLK_OUT output of the interface carries the read clock into the RAM.

Figure 4 • Simulation of the Test Procedure of the Example

ROM Emulation in ProASIC^{PLUS} Devices

The ROM emulation application in ProASIC^{PLUS} devices is based on RAM block initialization. If the users main design only has access to the read ports of the RAM block (RADDR, RD, RCLK, and REN) and the contents of the RAM are already initialized through the TAP, then the memory blocks is emulate ROM functionality for the core design. In this case, the write ports of the RAM blocks are only accessed by the user interface block and the interface is only activated by the TAP IR contents.

Users must note that the contents of the SRAM blocks are lost in the absence of applied power. During each power-up cycle, the initial data of the RAM must be loaded into the device through JTAG.

Conclusion

ProASIC^{PLUS} devices contain embedded dual-port RAM blocks. The blocks can be cascaded together to form deeper and wider memory blocks. These RAM blocks offer the flexibility of being initialized through JTAG pins. The initialization data is passed into the TAP and a user interface block is required to synchronize the initialization information.

The initialized memory blocks can be used in ROM emulation applications where the users design only reads from the memory blocks and the data remains intact. Users can reinitialize the RAM blocks or modify their contents through JTAG by applying the appropriate signals to the TAP of the device.

Appendix: Interface Block

```
`define Initialize_start 8'h22 //INITIALIZATION START COMMAND VALUE
`define Initialize_stop 8'h23 //INITIALIZATION START COMMAND VALUE

module interface(IR, rst_n, data_shift, clk_in, data_update, din_ser, dout_ser, test,

test_out,test_clk,clk_out,wr_en,rd_en,write_word,read_word,rd_addr,
wr_addr);

input [7:0] IR;
input [3:0] read_word; //RAM DATA READ BACK

input rst_n, data_shift, clk_in, data_update, din_ser; //INITIALIZATION SIGNALS
input test, test_clk; //TEST PROCEDURE CLOCK AND COMMAND INPUT

output [3:0] test_out; //READ DATA
output [3:0] write_word; //WRITE DATA
output [1:0] rd_addr; //READ ADDRESS
output [1:0] wr_addr; //WRITE ADDRESS

output dout_ser; //TDO DRIVER
output clk_out, wr_en, rd_en;

wire [3:0] write_word;
wire [1:0] rd_addr;
wire [1:0] wr_addr;
wire [3:0] Q_out;
wire enable, test_active;
reg clk_out;

//SELECT CLOCK FOR INITIALIZATION OR READBACK TEST

always @(enable or test_clk or data_update)
begin

case ({test_active})
1 : clk_out = test_clk ;
0 : clk_out = !data_update;
default : clk_out = 1'b1;
endcase

end

assign test_active = test && (IR == 8'h23);

assign enable = (IR == 8'h22);

assign wr_en = !enable;

assign rd_en = !test_active;

assign test_out = read_word;
assign dout_ser = Q_out[3];

//4-bit SIN/POUT SHIFT REGISTER
shift_reg data_shift_reg (.Shiften(data_shift), .Shiftin(din_ser), .Clock(clk_in),
.Q(Q_out));

//4-bit PIPELINE REGISTER
D_pipeline pipeline_reg (.Data(Q_out), .Clock(data_update), .Q(write_word));

//
addr_counter counter_1 (.Clock(data_update), .Q(wr_addr), .Aset(rst_n),
.Enable(enable));

addr_counter counter_2 (.Clock(test_clk), .Q(rd_addr), .Aset(rst_n),
.Enable(test_active));

endmodule
```

The Appendix of this document includes the Verilog code for the counter, shift register, pipeline register and the memory blocks.

The following is a sample wrapper, which connects the interface block to the UJTAG and the memory blocks:

```
// WRAPPER
module top_init (TDI, TRSTB, TMS, TCK, TDO, test, test_clk, test_out);
input TDI, TRSTB, TMS, TCK;
output TDO;
input test, test_clk;
output [3:0] test_out;

wire [7:0] IR;
wire reset, DR_shift, DR_cap, init_clk, DR_update, data_in, data_out;
wire clk_out, wen, ren;
wire [3:0] word_in, word_out;
wire [1:0] write_addr, read_addr;
UJTAG UJTAG_U1
(.UIREG0(IR[0]),.UIREG1(IR[1]),.UIREG2(IR[2]),.UIREG3(IR[3]),.UIREG4(IR[4]),

.UIREG5(IR[5]),.UIREG6(IR[6]),.UIREG7(IR[7]),.URSTB(reset),.UDRSH(DR_shift),.UDRCAP(DR
_cap),.UDRCK(init_clk),

.UDRUPD(DR_update),.UTDI(data_in),.TDI(TDI),.TMS(TMS),.TCK(TCK),
.TRSTB(TRSTB),.TDO(TDO),.UTDO(data_out));

mem_block RAM_block (.DO(word_out), .RCLOCK(clk_out), .WCLOCK(clk_out), .DI(word_in),
.WRB(wen),
.RDB(ren), .WADDR(write_addr), .RADDR(read_addr));

interface init_block (.IR(IR), .rst_n(reset), .data_shift(DR_shift),
.clk_in(init_clk),
.data_update(DR_update), .din_ser(data_in), .dout_ser(data_out), .test(test),
.test_out(test_out), .test_clk(test_clk), .clk_out(clk_out), .wr_en(wen),
.rd_en(ren), .write_word(word_in), .read_word(word_out), .rd_addr(read_addr),
.wr_addr(write_addr));

endmodule
```

Appendix: Address Counter

```
module addr_counter (Clock, Q, Aset, Enable);

input Clock;
output [1:0] Q;
input Aset;
input Enable;

reg [1:0] Qaux;

always @(posedge Clock or negedge Aset)
begin

if (!Aset)
Qaux <= 2'b11;
else if (Enable)
Qaux <= Qaux + 1;

end

assign Q = Qaux;

endmodule

Pipeline Register:
module D_pipeline (Data, Clock, Q);

input [3:0] Data;
input Clock;
output [3:0] Q;

reg [3:0] Q;
always @ (posedge Clock)
Q <= Data;

endmodule
```

Appendix: 4 × 4 RAM Block (Created by ACTgen Macro Builder)

```
module mem_block(DO, RCLOCK, WCLOCK, DI, WRB, RDB, WADDR, RADDR);
output [3:0] DO;
input RCLOCK;
input WCLOCK;
input [3:0] DI;
input WRB;
input RDB;
input [1:0] WADDR;
input [1:0] RADDR;

GND U1(.Y(VSS));
RAM256x9SSR M0(.RCLKS(RCLOCK), .WCLKS(WCLOCK), .DO8(), .DO7(), .DO6(), .DO5(), .DO4(),
.DO3(DO[3]), .DO2(DO[2]), .DO1(DO[1]), .DO0(DO[0]), .DOS(), .WPE(), .RPE(),
.WADDR7(VSS), .WADDR6(VSS),
.WADDR5(VSS), .WADDR4(VSS), .WADDR3(VSS), .WADDR2(VSS), .WADDR1(WADDR[1]),
.WADDR0(WADDR[0]),
.RADDR7(VSS), .RADDR6(VSS), .RADDR5(VSS), .RADDR4(VSS), .RADDR3(VSS), .RADDR2(VSS),
.RADDR1(RADDR[1]), .RADDR0(RADDR[0]), .DI8(VSS), .DI7(VSS), .DI6(VSS), .DI5(VSS),
.DI4(VSS),
.DI3(DI[3]), .DI2(DI[2]), .DI1(DI[1]), .DI0(DI[0]), .WRB(WRB), .RDB(RDB), .WBLKB(VSS),
.RBLKB(VSS),
.PARODD(VSS), .DIS(VSS));
endmodule
```

List of Changes

The following table shows important changes made in this document for each revision.

Revision	**Changes**	**Page**
Revision 1 (June 2016)	Non-technical updates.	N/A
Revision 0 (December 2002)	Initial release.	N/A

Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
E-mail: sales.support@microsemi.com
www.microsemi.com

About Microsemi

5190005-1/6.16