Using the JTAG Interface to the fido1100

An Innovasic Semiconductor Application Note
fido1100 Application Note 170

Version 1.1

May 2007

Innovasic Semiconductor

Table of Contents

List of Tables

List of Figures

Abstract

This Application Note describes the JTAG interface of the fido1100 and details its implementation and functionality. Knowledge of the JTAG 1149.1-2001 IEEE specification, boundary-scan architecture, and Boundary-Scan Description Language (BSDL) is recommended prior to implementing your own interface. For additional information refer to IEEE Standard 1149.1-2001, titled IEEE Standard Test Access Port and Boundary-Scan Architecture.

The Application Note is written for both users of the fido1100 development board and users of the fido100 chip. Circuits are suggested for use with the fido1100 chip and hardware is listed for use with the fido1100 development board.

Approach

Introduction

This application note describes how the fido1100 JTAG interface operates at two levels. The first level shows how the JTAG interface operates within the fido1100 chip and the second level discusses how Innovasic Semiconductor has implemented this interface on the fido1100 EDK development board.

Chip

The fido1100 chip is fully compliant with the IEEE 1149.1-2001 Test Access Port and Boundary-Scan Architecture. The fido1100 architecture is equipped with the Test Access Port (TAP), TAP controller, instruction register, instruction decoder, boundary-scan register, and by-pass register. The chip has 4 pins that conform to the JTAG standard to provide chip level testing. In addition, the fido1100 has extra registers that provide debug capabilities beyond the JTAG standard.

Board

The fido1100 development board supplied in the EDK provides board level testing through two JTAG interfaces: a USB interface that requires a simple USB cable or an ARM20 Header that requires a Parallel/USB wiggler. The board is loaded with firmware from Macraigor Systems (www.macraigor.com) and integrates several software tools.

JTAG Overview

JTAG Pin Architecture

The fido1100 uses the minimum number of pins required by the JTAG IEEE standard. In Table 1the four signals of the JTAG interface are described in detail.

Table 1: JTAG Interface Signals

Pin	Direction	Description
TDO	Out	**Test Data Output** - The tri-state test data output changing on the falling edge of the TCK input. This is actively driven only in the shift-DR and shift-IR controller states (see Figure 1).
TDI	In	**Test Data Input** - The test data input sampled on the rising edge of the TCK input.
TMS	In	**Test Mode Select Input** – The test mode select input used to sequence the TAP controller state machine (see Figure 1).
TCK	In	**Test Clock Input** - All JTAG commands and serial data are synchronized by this signal.

NOTE: There is no JTAG Rest Signal (TRST). The reset block is discussed in the fido1100 hardware chip reset section (JTAG Reset, page 9).

JTAG Port Architecture, TAP controller and Timing Signals

The TAP (Test Access Port) Controller is a 16-state state machine used to sequence the JTAG port by:

1. Serially shifting JTAG port instructions in or out and decoding them.
2. Serially input or output a data value.
3. Update the JTAG port register.

The TAP controller is a synchronous finite state machine and responds to changes in the TMS and TCK signals. States transition on the rising edge of TCK. Values shown to the side of each state represent the state of TMS at the time of the rising edge of TCK.

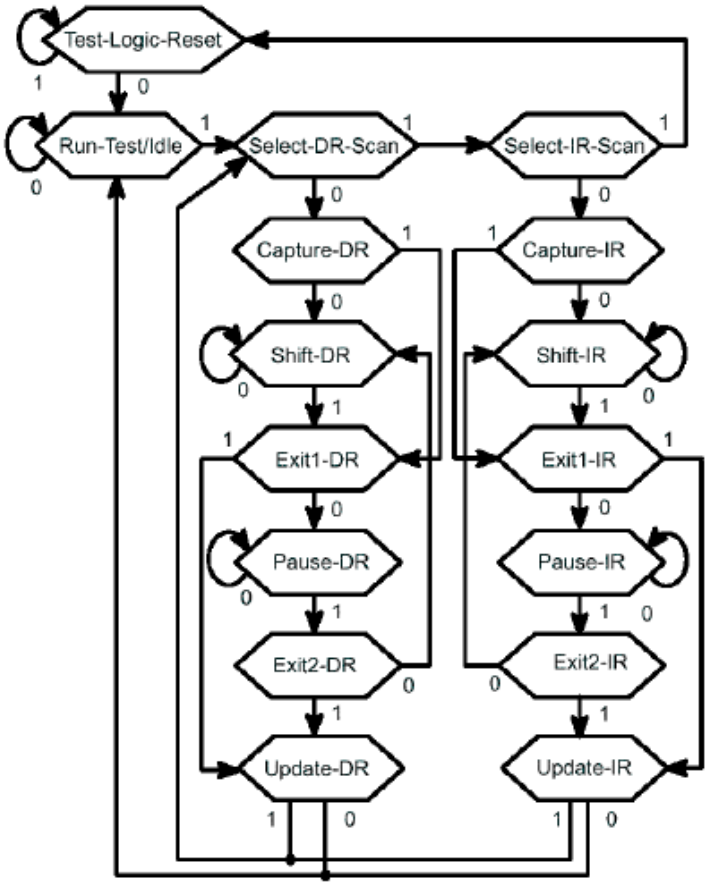

Figure 1: TAP Controller Finite State Machine

There are two paths through the state machine: the instruction path and the data path. The instruction path captures and loads the JTAG instructions into the instruction register. The data path captures and loads data into the data registers. The TAP controller executes the last instruction decode until a new instruction is entered at the Update-IR state or until a reset is sent to the controller.

The JTAG port has multiple Read/Write registers. An ID register, By-Pass Register, Boundary Scan and Instruction Register. These registers are complemented with additional registers to enable debug functionality in the fido1100.

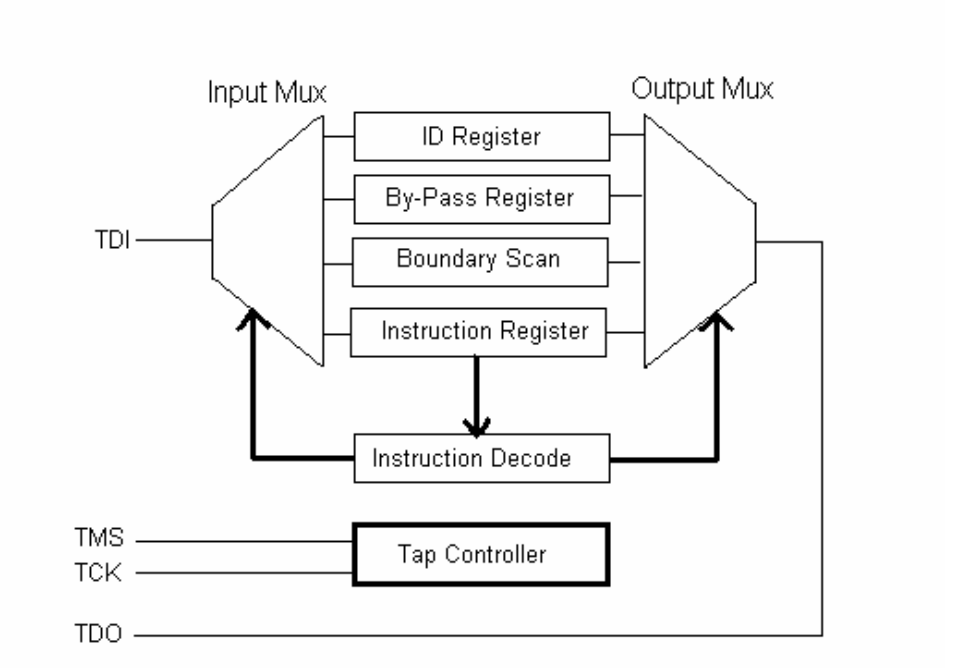

Figure 2: JTAG Port Signal Interface

The timing of the JTAG signals is shown below. The TDO pin remains in the high impedance state except during a shift-DR or shift-IR controller state. In the shift-DR and shift-IR controller states, TDO is updated on the falling edge of TCK. TMS and TDI are sampled on the rising edge of TCK.

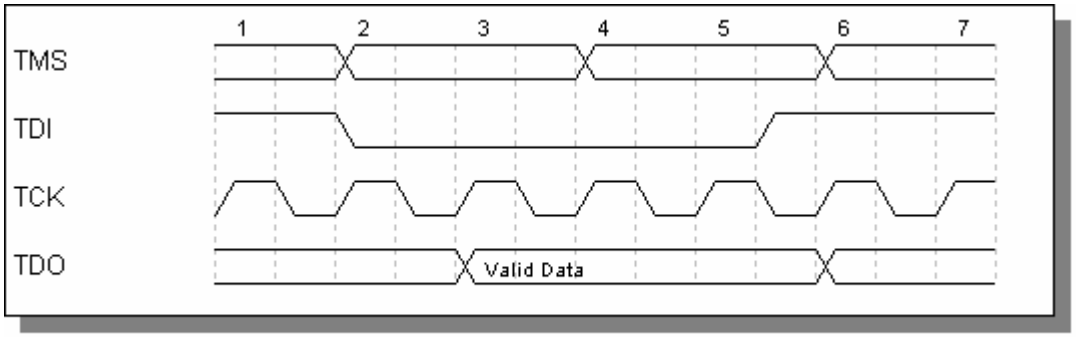

Figure 3: JTAG Interface Timing Diagram

JTAG Interface Specifications for the fido1100

The JTAG interface is supported in both the Ball Grid Array (BGA) and flat pack packages. Table 2 lists the JTAG interface signals and respective pins one each package.

Table 2: JTAG Interface Pin Outs

Signal Name	Direction	PQFP Pin	BGA Pin
TDI	Input	102	T16
TDO	Output	103	U16
TCK	Input	104	V17
TMS	Input	105	U17

JTAG Reset

On power-up, the Power On Reset (POR) cell, *internal to the chip*, will reset the JTAG block. The JTAG block can also reset the fido1100 CORE, but only the POR cell can reset the JTAG block.

Reset is achieved by pulling the POR pin high. The POR is tied to the JTAG block and OR-ed to the other reset lines (see Figure 4). In reset the Time Mode Select (TMS) input = 1.

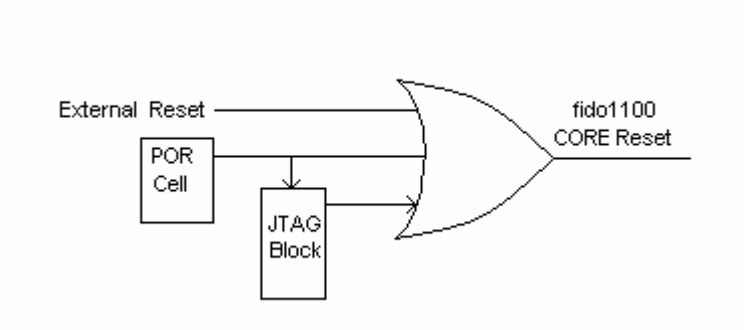

Figure 4: JTAG Reset Operation

JTAG Transmission Rates

It is best to initially set-up your system with a low JTAG clock frequency, such as 1 MHz. Once you have established communication with the target, you can increase the JTAG clock frequency to the highest frequency that maintains stable, consistent operation of the emulator and debugger. The frequency selected should not exceed one fourth (1/4) the frequency of the 66MHz system clock.

JTAG Scan Chain and Debug Functionality

The JTAG port contains an 8-bit wide instruction register. Instructions are transferred to this register during the shift-IR state of the TAP state machine and are decoded by entering the Update-IR state of the TAP state machine (see Figure 1). The JTAG controller executes the last decoded instruction until another new instruction is entered and decoded. The instructions and data are entered serially through the TDI pin, LSB first.

The JTAG Test Access Port (TAP) instruction shift register will support the following debug scan chain commands:

Table 3: JTAG Scan Chain Functions

JTAG Instruction	Scan Chain Function	Scan Chain Length	Scan Chain Reference Number	Public or Private
00010000	**READWRITEADDRCMD** (Read/Write Memory/Registers Address and Command)	37 bits	1	Private
00010001	**READDATA** (Read Memory/Registers Data)	32 bits	2	Private
00010010	**WRITEDATA** (Write Memory/Registers Data)	32 bits	7	Private
00010011	**READPC_ANDCONTEXT** (Read Program Counter and Active context)	37 bits	4	Private
00010100	**READWRITEDRBUGREG** (Read/Write Debug Control Register)	15 bits	5	Private
11111110	**IDCODE** (Read Device ID Register)	32 bits	3	Public
11111000	**EXTEST** (IO Boundary Scan)	346 bits	6	Public
11111010	**SAMPLE/PRELOAD** (Sample Boundary Scan chain on 'Capture-DR' state, Load Boundary Scan chain on 'Update-DR' state)	346 bits	6	Public
11111111	**BYPASS** (Use TDI/TDO Bypass Register)	1 bit	9	Public
00000111	**RUNBIST** (Run Built in Self Test)	16 bits	8	Public
00001111	**ENABLEATPG** (Enable ATPG Mode for Manufacturing Test ONLY) see NOTE 5	N/A	N/A	Private

1.) The boundary-scan scan chain is selected via the **EXETEST**, **SAMPLE**, and **PRELOAD** instructions.
2.) The **SAMPLE** and **PRELOAD** instructions have the SAME binary code. (They are identified as separate instructions in the JTAG Spec, but are allowed to have the same binary code for backwards compatibility with previous version of spec).
3.) Any undefined bit pattern that is shifted into the Instruction Register will perform the same function as the **BYPASS** instruction.
4.) On Power-on Reset, or when the JTAG state machine enters the 'Test Logic Reset' the instruction register will reset its value to operate as the **IDCODE** Instruction (per JTAG Spec).
5.) The **ENABLEATPG** instruction must not be used. It exists for manufacture testing only.

Private Instructions

READWRITEADDRCMD -- Selects the Read/Write Memory/Registers Address Scan Chain to be written to (Update-DR state). This register can also be read (Capture-DR State).

A. To read a memory/register:

1. Set the IR to the READWRITEADDRCMD command (address and read mode)
2. Write the address (and control data) to the Read/Write Memory/Registers Address Scan Chain (1)
    - The Address and read command Data will be shifted into the Scan chain
    - The JTAG Machine will copy the scan chain to the Address Register on the Update-DR state
    - The Execution unit will start a read (using the Address Register)
    - The results of the read will go into the read data register (when it is finished)

3. Set the IR to the READDATA command.

4. Read the data from the Read Memory/Registers Data Scan Chain (2)

    - The JTAG machine will copy the read data register into the scan chain as soon as the Capture-DR state is entered

        - The Address Register will be incremented (if necessary, per the READWRITEADDRCMD command)
        - The Execution unit will automatically start another read (using the updated address register, this should happen close to the Capture-DR state)

5. The resulting Read data will be shifted out of the scan chain

NOTE: The Test system can perform another read by just reentering the Capture-DR state again.

NOTE: The data shifted out (read) from the Read Memory/Registers Data Scan Chain will actually be the result of the previous read. It is possible that stale data could be returned if time between reads is significant. To ensure fresh data, read the data twice, or select the address again and then read.

B. To write a memory/register:

1. Set the IR to the READWRITEADDRCMD command (address and write mode)
2. Write the address (and control data) to the Read/Write Memory/Registers Address Scan Chain (1)
    - The Address and write command Data will be shifted into the Scan chain
    - The JTAG Machine will copy the scan chain to the Address Register on the Update-DR state

3. Set the IR to the WRITEDATA command.

4. Write the data to the Write Memory/Registers Data Scan Chain (7)

- The Data to be written will be shifted into the Scan chain
- The JTAG Machine will copy the scan chain to the Write Data Register on the Update-DR state
- The Execution unit will start a write (of the Write Data Register and Address Register) as soon as Update-DR state is entered
- The Address Register will be incremented (if necessary, per the READWRITEADDRCMD command)

NOTE: The Test system can perform another write by just reentering the Capture-DR state, shifting new write data, and entering the Update-DR state again.

READDATA -- Selects the Read Memory/Registers Data Scan Chain to be read. This register is read only.

WRITEDATA -- Selects the Write Memory/Registers Data Scan Chain to be written to.

READPC ANDCONTEXT -- Selects the Program Counter and Active Context Scan Chain to be read. This register is read only.

READWRITEDEBUGREG -- Selects the Debug control register Scan Chain to be read and written to.

ENABLEATPG -- Will put the chip into ATPG Manufacturing test mode, to leave this mode, a different instruction must be shifted into the IR or a Reset must occur. This mode is for reserved for internal purposes only.

Public Instructions

EXTEST -- The **EXTEST** instruction selects the boundary-scan scan chain, (see Pin out and Boundary Scan) and allows the execution of the interconnections test, i.e. the check of the connection between the output of a circuit and the input of another circuit. This instruction will interrupt the connection between the pins and the on-chip core logic.

SAMPLE -- Data is loaded into the Boundary Scan register (sample). A snapshot of the signals on the pins of the component is taken. This instruction has no effect on the on-chip system logic (i.e. pins are connected to core logic).

PRELOAD -- Known data patterns are loaded into the Boundary-Scan register output cells (preload). A pattern of data can be loaded onto the Boundary Scan register before it is driven out on the pins (via an EXTEST instruction). This instruction has no effect on the on-chip system logic (i.e. pins are connected to core logic).

IDCODE -- Return the Device ID and version register.

BYPASS -- The **BYPASS** instruction switches the bypass register into the TDI/TDO path

RUNBIST -- Runs the Built in Self Test, and the results are returned in the BIST Results Register Scan Chain. The Self-Test will only run when the JTAG state machine is in the Run Test/Idle state.

PUBLIC INSTRUCTIONS NOT IMPLEMENTED (FOR REFERENCE ONLY)

INTEST -- Apply the Boundary Scan Register to the Internal Core Logic of the part.

CLAMP -- Place a 'guarding' signal to be applied to the Boundary Scan Register.

HIGHZ -- Place all system logic outputs to inactive drive state.

USERCODE -- Similar to **IDCODE,** however only for programmable devices.

JTAG SCAN CHAINS

Scan Chain 1: Read/Write Memory/Registers Address Scan Chain READ/WRITE

36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mode			Command		Address																																	

1. 37 Bits in length:
2. Bits 36-34:
    a. Mode:
        - 000 -> Read/Write Byte (8 bits) Data From Memory Address (internal / external memory or registers).
        - 001 -> Read/Write Word (16 bits) Data From Memory Address (internal / external memory or registers).
        - 010 -> Read/Write Long (32 bits) Data from Memory Address (internal / external memory or registers).
        - 011 -> Read/Write Long (32 bits) Data From Non-Memory mapped registers
        - 100 -> Read/Write Long (32 bits) Data From offset memory.
        - 101 -> Reserved
        - 110 -> Reserved
        - 111 -> Reserved
3. Bits 33-32:
    a. Command:
        - 00 -> read
        - 01 -> read and increment address

- 10 -> write
- 11 -> write and increment address

4. Bits 31-0:
    a. Memory/Register Address

Scan Chain 2: Read Memory/Registers Data Scan Chain – READ ONLY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA																															

1. 32 Bits in length:
    - Bits 31-0:
        - Memory/Register Data

Scan Chain 3: Write Memory/Registers Data Scan Chain – WRITE ONLY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA																															

1. 32 Bits in length:
    - Bits 31-0:
        - Memory/Register Data

Scan Chain 4: Read Program Counter and Active context Scan Chain – READ ONLY

36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			Active Context		Program Counter																																	

1. 37 Bits in length:
    - Bits 36-32:
        - Active Context, 0-7, upper two bits reserved.
    - Bits 31-0:
        - Program Counter Data.

Scan Chain 5: Read Device ID Register Scan Chain – READ ONLY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Version				Part Number																Manufacturer ID											LSB

1. 32 Bits in length:
    - Bits 31-28 (4 bits):
        i. Version
    - Bits 27-12 (16 bits):
        i. Part Number
    - Bits 11-1 (11 bits):
        i. Manufacturer
    - Bit 0:
        i. LSB = 0x1 (Hard coded to 1 by JTAG Spec)

2. Register contents == 0x01100531 for fido1100

Scan Chain 6: Read/Write Debug Control Register Scan Chain – READ/WRITE

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write/Poll	Single Step					Halt Command					Freeze WDT	Reserved	Reset	Break Point Mode

1. Provides these control functions
    - Start
    - Stop
    - Resume
    - Single Step
    - Resurrect
    - Reset
    - Query for CPU state
2. 15 Bits in length:
    - Bit 14:
        a. If set will write the bits into the debug register (halt, etc), if cleared will just poll for status.
        b. This bit will always read as a 0.
    - Bits 13-9:
        a. Single Step Command/Status Bits, 1 bit per context. Self clearing after single instruction completes.
        b. Bit 13 = Context 4, bit 9 = context 0

c. Context must be in Halted state (bits 8-4 set) for bit to have any effect. If Single Step is reading for context, read = 1, if step is complete read = 0.

- Bits 8-4:
  a. Halt Command/Status Bits, 1 bit per context.
  b. Bit 8 = Context 4, bit 4 = context 0
  c. Setting bit via JTAG will immediately halt that context (after the current instruction)
  d. If the context hits a breakpoint that bit will be set and can be polled by the debugger.
- Bit 3:
  a. Freeze Watchdog Timer - bit is latched in the scan chain and WDT will remain frozen until released.
- Bit 2:
  a. Reserved.
- Bit 1:
  a. Reset the CPU (command only, same as the reset input signal, bit is latched in the scan chain and CPU will be held in reset until released)
- Bit 0:
  a. Break Point Mode (command only)
    - 0 -> Standard (exception based) breakpoints
    - 1 -> JTAG controlled breakpoints.

Scan Chain 7: BIST Results Register Scan Chain – READ ONLY

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								D Cache RAM Done	User RAM Done	TX Done	Rx Done	D Cache RAM Fail	User RAM Fail	TX Fail	RX Fail

1. Bit 15-8 - Reserved
2. Bit 7
   D Cache Done - The BIST sequence for the Deterministic Cache RAM block has completed (this one should take the longest)
3. Bit 6
   User RAM Done - The BIST sequence for the User RAM block has completed
4. Bit 5
   TX Done - The BIST sequence for the PDMA TX FIFO (dual port RAM) block has completed
5. Bit 4
   RX Done - The BIST sequence for the PDMA RX FIFO (dual port RAM) block has completed
6. Bit 3
   D Cache RAM Fail - Bist has failed for the Deterministic Cache RAM block (a pass (logic 0) is only valid when the corresponding Done flag is set)
7. Bit 2
   User RAM Fail - Bist has failed for the User RAM block (a pass (logic 0) is only valid when the corresponding Done flag is set)
8. Bit 1
   TX Fail - Bist has failed for the PDMA TX FIFO block (a pass (logic 0) is only valid when the corresponding Done flag is set)
9. Bit 0
   RX Fail - Bist has failed for the PDMA RX FIFO block (a pass (logic 0) is only valid when the corresponding Done flag is set)

Notes:

1. This register will identify the results of the BIST
2. BIST will be implemented by the **RUNBIST** instruction, and will be executed only when the JTAG State Machine is in the Run Test/Idle state
3. The test will run a predetermined amount of time (number of tck's) from when the JTAG State Machine enters the Run Test/Idle state.
4. The JTAG state machine must remain in the Run Test/Idle state until this time elapses.
5. The results must be latched into the BIST Results Register Scan Chain when it is finished.
6. If memory fails, the fail flag for that memory will be set, and the done flag will be clear.
7. If memory passes the fail flag for that memory will be clear, and the done flag will be set.

Scan Chain 8: I/O BOUNDARY Scan Chain – READ ONLY

The JTAG BSR (Boundary Scan Register) is a large register that contains the signal information and control for external signals. Each cell (bit) in the register controls one aspect of an external signal. External signals may have 1, 2, or 3 internal cells that correspond to its control.

1. The BSR is controlled by the JTAG Instructions:

    - **SAMPLE/PRELOAD**
        i. Capture a copy of the external signals onto the BSR when JTAG enters the Capture-DR state (SAMPLE) so that it can be shifted out.
        ii. Load the BSR from the shift register when JTAG enters the Update-DR state (PRELOAD).
    - **EXTEST**
        i. Drive the BSR onto the external signals

2. On the signals below:

    - _in identifies the value of the input signal from the external signal.
    - _out identifies the value of the output signal to be driven onto the external signal.
    - _en identifies if the output signal driver is to drive the signal (_out value) onto the external signal or tristate the driver. (1 = drive, 0 = tristate.)

BSR Scan Chain Register

Table 4: BSR Scan Chain Register

Bit	Signal Name	Notes
345	uic0_0_in	
344	uic0_0_out	
343	uic0_0_en	
342	uic0_1_in	
341	uic0_1_out	
340	uic0_1_en	
339	uic0_2_in	
338	uic0_2_out	
337	uic0_2_en	
336	uic0_3_in	
335	uic0_3_out	
334	uic0_3_en	
333	uic0_4_in	
332	uic0_4_out	
331	uic0_4_en	
330	uic0_5_in	
329	uic0_5_out	
328	uic0_5_en	
327	uic0_6_in	
326	uic0_6_out	
325	uic0_6_en	
324	uic0_7_in	
323	uic0_7_out	
322	uic0_7_en	
321	uic0_8_in	
320	uic0_8_out	
319	uic0_8_en	
318	xtal_0_in	
317	uic0_9_in	
316	uic0_9_out	
315	uic0_9_en	
314	uic0_10_in	
313	uic0_10_out	
312	uic0_10_en	
311	uic0_11_in	
310	uic0_11_out	
309	uic0_11_en	
308	uic0_12_in	
307	uic0_12_out	
306	uic0_12_en	
305	uic0_13_in	
304	uic0_13_out	
303	uic0_13_en	

Bit	Signal Name	Notes
302	uic0_14_in	
301	uic0_14_out	
300	uic0_14_en	
299	uic0_15_in	
298	uic0_15_out	
297	uic0_15_en	
296	uic0_16_in	
295	uic0_16_out	
294	uic0_16_en	
293	uic0_17_in	
292	uic0_17_out	
291	uic0_17_en	
290	uic1_0_in	
289	uic1_0_out	
288	uic1_0_en	
287	uic1_1_in	
286	uic1_1_out	
285	uic1_1_en	
284	uic1_2_in	
283	uic1_2_out	
282	uic1_2_en	
281	uic1_3_in	
280	uic1_3_out	
279	uic1_3_en	
278	uic1_4_in	
277	uic1_4_out	
276	uic1_4_en	
275	uic1_5_in	
274	uic1_5_out	
273	uic1_5_en	
272	uic1_6_in	
271	uic1_6_out	
270	uic1_6_en	
269	uic1_7_in	
268	uic1_7_out	
267	uic1_7_en	
266	uic1_8_in	
265	uic1_8_out	
264	uic1_8_en	
263	uic1_9_in	
262	uic1_9_out	
261	uic1_9_en	
260	uic1_10_in	
259	uic1_10_out	
258	uic1_10_en	
257	uic1_11_in	

Bit	Signal Name	Notes
256	uic1_11_out	
255	uic1_11_en	
254	uic1_12_in	
253	uic1_12_out	
252	uic1_12_en	
251	uic1_13_in	
250	uic1_13_out	
249	uic1_13_en	
248	uic1_14_in	
247	uic1_14_out	
246	uic1_14_en	
245	uic1_15_in	
244	uic1_15_out	
243	uic1_15_en	
242	uic1_16_in	
241	uic1_16_out	
240	uic1_16_en	
239	uic1_17_in	
238	uic1_17_out	
237	uic1_17_en	
236	uic2_0_in	
235	uic2_0_out	
234	uic2_0_en	
233	uic2_1_in	
232	uic2_1_out	
231	uic2_1_en	
230	uic2_2_in	
229	uic2_2_out	
228	uic2_2_en	
227	uic2_3_in	
226	uic2_3_out	
225	uic2_3_en	
224	uic2_4_in	
223	uic2_4_out	
222	uic2_4_en	
221	uic2_5_in	
220	uic2_5_out	
219	uic2_5_en	
218	uic2_6_in	
217	uic2_6_out	
216	uic2_6_en	
215	uic2_7_in	
214	uic2_7_out	
213	uic2_7_en	
212	uic2_8_in	
211	uic2_8_out	

Bit	Signal Name	Notes
210	uic2_8_en	
209	uic2_9_in	
208	uic2_9_out	
207	uic2_9_en	
206	uic2_10_in	
205	uic2_10_out	
204	uic2_10_en	
203	uic2_11_in	
202	uic2_11_out	
201	uic2_11_en	
200	uic2_12_in	
199	uic2_12_out	
198	uic2_12_en	
197	uic2_13_in	
196	uic2_13_out	
195	uic2_13_en	
194	uic2_14_in	
193	uic2_14_out	
192	uic2_14_en	
191	uic2_15_in	
190	uic2_15_out	
189	uic2_15_en	
188	uic2_16_in	
187	uic2_16_out	
186	uic2_16_en	
185	uic2_17_in	
184	uic2_17_out	
183	uic2_17_en	
182	uic3_0_in	
181	uic3_0_out	
180	uic3_0_en	
179	uic3_1_in	
178	uic3_1_out	
177	uic3_1_en	
176	uic3_2_in	
175	uic3_2_out	
174	uic3_2_en	
173	uic3_3_in	
172	uic3_3_out	
171	uic3_3_en	
170	uic3_4_in	
169	uic3_4_out	
168	uic3_4_en	
167	uic3_5_in	
166	uic3_5_out	
165	uic3_5_en	

Bit	Signal Name	Notes
164	uic3_6_in	
163	uic3_6_out	
162	uic3_6_en	
161	uic3_7_in	
160	uic3_7_out	
159	uic3_7_en	
158	uic3_8_in	
157	uic3_8_out	
156	uic3_8_en	
155	uic3_9_in	
154	uic3_9_out	
153	uic3_9_en	
152	uic3_10_in	
151	uic3_10_out	
150	uic3_10_en	
149	uic3_11_in	
148	uic3_11_out	
147	uic3_11_en	
146	uic3_12_in	
145	uic3_12_out	
144	uic3_12_en	
143	uic3_13_in	
142	uic3_13_out	
141	uic3_13_en	
140	uic3_14_in	
139	uic3_14_out	
138	uic3_14_en	
137	uic3_15_in	
136	uic3_15_out	
135	uic3_15_en	
134	uic3_16_in	
133	uic3_16_out	
132	uic3_16_en	
131	uic3_17_in	
130	uic3_17_out	
129	uic3_17_en	
128	t0ic0_t0oc0_in	
127	t0ic0_t0oc0_out	
126	t0ic0_t0oc0_en	
125	t0ic1_t0oc1_in	
124	t0ic1_t0oc1_out	
123	t0ic1_t0oc1_en	
122	t0ic2_t0oc2_in	
121	t0ic2_t0oc2_out	
120	t0ic2_t0oc2_en	
119	t0ic3_t0oc3_in	

Bit	Signal Name	Notes
118	t0ic3_t0oc3_out	
117	t0ic3_t0oc3_en	
116	t1ic0_t1oc0_in	
115	t1ic0_t1oc0_out	
114	t1ic0_t1oc0_en	
113	t1ic1_t1oc1_in	
112	t1ic1_t1oc1_out	
111	t1ic1_t1oc1_en	
110	t1ic2_t1oc2_in	
109	t1ic2_t1oc2_out	
108	t1ic2_t1oc2_en	
107	t1ic3_t1oc3_in	
106	t1ic3_t1oc3_out	
105	t1ic3_t1oc3_en	
104	t0in_in	
103	t1in_in	
102	int0_in	
101	int1_in	
100	int2_in	
99	int3_in	
98	int4_dma0_ack_in	
97	int4_dma0_ack_out	
96	int4_dma0_ack_en	
95	int5_dma1_ack_in	
94	int5_dma1_ack_out	
93	int5_dma1_ack_en	
92	int6_dma0_req_in	
91	int7_dma1_req_in	
90	d_en	
89	d0_in	
88	d0_out	1
87	d1_in	
86	d1_out	1
85	d2_in	
84	d2_out	1
83	d3_in	
82	d3_out	1
81	d4_in	
80	d4_out	1
79	d5_in	
78	d5_out	1
77	d6_in	
76	d6_out	1
75	d7_in	
74	d7_out	1
73	d8_in	

Bit	Signal Name	Notes
72	d8_out	1
71	d9_in	
70	d9_out	1
69	d10_in	
68	d10_out	1
67	d11_in	
66	d11_out	1
65	d12_in	
64	d12_out	1
63	d13_in	
62	d13_out	1
61	d14_in	
60	d14_out	1
59	d15_in	
58	d15_out	1
57	rdy_n_in	
56	memclk_out	2
55	ext_ctl_en	
54	be0_n_out	2
53	be1_n_out	2
52	oe_n_out	2
51	rw_n_out	2
50	ba0_out	2
49	ba1_out	2
48	cas_n_out	2
47	ras_n_out	2
46	cke_out	2
45	holdreq_n_in	
44	holdgnt_n_out	
43	reset_in	
42	reset_out_out	
41	ext_addr_en	
40	a0_out	3
39	a1_out	3
38	a2_out	3
37	a3_out	3
36	a4_out	3
35	a5_out	3
34	a6_out	3
33	a7_out	3
32	a8_out	3
31	a9_out	3
30	a10_out	3
29	a11_out	3
28	a12_out	3
27	a13_out	3

Bit	Signal Name	Notes
26	a14_out	3
25	a15_out	3
24	a16_out	3
23	a17_out	3
22	a18_out	3
21	a19_out	3
20	a20_out	3
19	a21_out	3
18	a22_out	3
17	a23_out	3
16	a24_out	3
15	reset_delay_in	
14	a25_out	3
13	size_in	
12	a26_out	3
11	a27_cs7_n_out	
10	a27_cs7_n_en	
9	a28_cs6_n_out	
8	a28_cs6_n_en	
7	a29_cs5_n_out	
6	a29_cs5_n_en	
5	a30_cs4_n_out	
4	a30_cs4_n_en	
3	cs0_n_out	2
2	cs1_n_out	2
1	cs2_n_out	2
0	cs3_n_out	2

Notes:

- Note 1: Output Signal controlled by d_en (bit 90)
- Note 2 Output Signal controlled by ext_ctl_en (bit 55)
- Note 3: Output Signal controlled by ext_addr_en (bit 41)

External Signals Controlled by BSR

Table 5: External Signals Controlled by BSR

Number	Signal Name
156	ui0_0
155	ui0_1
154	ui0_2
153	ui0_3
152	ui0_4
151	ui0_5
150	ui0_6
149	ui0_7
148	ui0_8
147	xtal0
146	ui0_9
145	ui0_10
144	ui0_11
143	ui0_12
142	ui0_13
141	ui0_14
140	ui0_15
139	ui0_16
138	ui0_17
137	ui1_0
136	ui1_1
135	ui1_2
134	ui1_3
133	ui1_4
132	ui1_5
131	ui1_6
130	ui1_7
129	ui1_8
128	ui1_9
127	ui1_10
126	ui1_11
125	ui1_12
124	ui1_13
123	ui1_14
122	ui1_15
121	ui1_16
120	ui1_17
119	ui2_0
118	ui2_1
117	ui2_2
116	ui2_3
115	ui2_4

Number	Signal Name
114	ui2_5
113	ui2_6
112	ui2_7
111	ui2_8
110	ui2_9
109	ui2_10
108	ui2_11
107	ui2_12
106	ui2_13
105	ui2_14
104	ui2_15
103	ui2_16
102	ui2_17
101	ui3_0
100	ui3_1
99	ui3_2
98	ui3_3
97	ui3_4
96	ui3_5
95	ui3_6
94	ui3_7
93	ui3_8
92	ui3_9
91	ui3_10
90	ui3_11
89	ui3_12
88	ui3_13
87	ui3_14
86	ui3_15
85	ui3_16
84	ui3_17
83	t0ic0_t0oc0
82	t0ic1_t0oc1
81	t0ic2_t0oc2
80	t0ic3_t0oc3
79	t1ic0_t1oc0
78	t1ic1_t1oc1
77	t1ic2_t1oc2
76	t1ic3_t1oc3
75	t0in
74	t1in
73	int0
72	int1
71	int2
70	int3
69	int4_dma0_ack

Number	Signal Name
68	int5_dma1_ack
67	int6_dma0_req
66	int7_dma1_req
65	d0
64	d1
63	d2
62	d3
61	d4
60	d5
59	d6
58	d7
57	d8
56	d9
55	d10
54	d11
53	d12
52	d13
51	d14
50	d15
49	rdy_n
48	Memclk
47	be0_n
46	be1_n
45	oe_n
44	rw_n
43	ba0
42	ba1
41	cas_n
40	ras_n
39	Cke
38	holdreq_n
37	holdgnt_n
36	Reset
35	reset_out
34	a0
33	a1
32	a2
32	a3
30	a4
29	a5
28	a6
27	a7
26	a8
25	a9
24	a10
23	a11

Number	Signal Name
22	a12
21	a13
20	a14
19	a15
18	a16
17	a17
16	a18
15	a19
14	a20
13	a21
12	a22
11	a23
10	a24
9	a25_reset_delay
8	a26_size
7	a27_cs7_n
6	a28_cs6_n
5	a29_cs5_n
4	a30_cs4_n
3	cs0_n
2	cs1_n
1	cs2_n
0	cs3_n

Hardware and Software Integration

The fido1100 chip allows applications to be integrated from a wide variety of existing environments. The integration of software and hardware tools is only limited by the designer's imagination with the caveat that the software must be capable of handling the private and public instructions of the fido1100. Examples of hardware/software integration are GDB debugging tools and BSDL testers.

Macraigor Overview

Innovasic has chosen to partner with Macraigor Systems for our fido1100 development board. Macraigor provides Innovasic with a low cost solution for its development board. In addition, Macraigor has a selection of hardware and software tools that augment the development board. These tools are system tested and verified to work with the fido1100 development board. While the user is free to use any tools they want, the testing and verification of such tools would be the user's responsibility.

Macraigor Hardware Requirements

To access the ARM20 Header interface on the fido1100 development board, a USB or parallel port wiggler is required. The wiggler is a very simple device that primarily acts as an interface buffer between the software and target JTAG signals.

The fido1100 has verified compatibility with 2 Macraigor Systems wigglers. It is highly recommended that you use the USB interface for your JTAG testing needs. The speed of the USB interface is superior to that of the parallel one.

Table 6: JTAG Interface Devices

Approved Wigglers	
Name	**Interface**
Wiggler	Parallel
usbWiggler	USB

NOTE: When ordering the wigglers, you must specify an ARM20 Header.

Macraigor Software Integration

The fido1100 system uses 2 different software tools provided by Macraigor: the ODC Remote and the GDB debugger.

1. **OCD Remote** – OCD Remote is a utility that listens on a TCP/IP port and translates GDB monitor commands into: Wiggler, USB Wiggler commands. It should be noted that OCD Remote is actually provided by Code Sourcery and links in Macraigor libraries.

2. **GDB -** GDB, the GNU Project debugger, allows you to see what is going on inside another program while it executes -- or what another program was doing at the moment it crashed. GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

    - Start your program, specifying anything that might affect its behavior.
    - Make your program stop on specified conditions.
    - Examine what has happened, when your program has stopped.
    - Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

JTAG Implementation on the fido1100 EDK development Board

The fido1100 development board has two interface options. The USB interface requires only a USB cable to connect to the host PC. Onboard firmware provided by Macraigor along with other hardware combine to create an on-board USB wiggler interface . The speed of the USB interface is 4.5 KHz to 18MHz.

The ARM20 Header interface requires an external wiggler. The wiggler can be either a USB wiggler or a parallel port wiggler. The speed of the parallel wiggler is 60 KHz to 380KHz. The USB wiggler can top out at 480mb/s. Both interfaces will require the software mentioned above on the host PC.

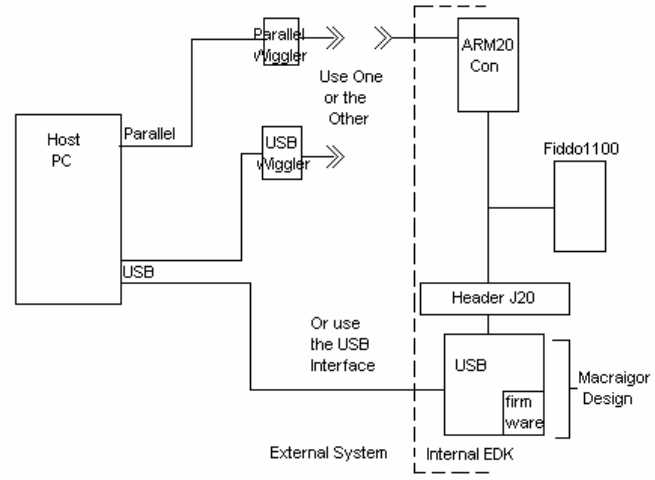

Figure 5: JTAG Interface Options on the fodo1100 EDK Development Board

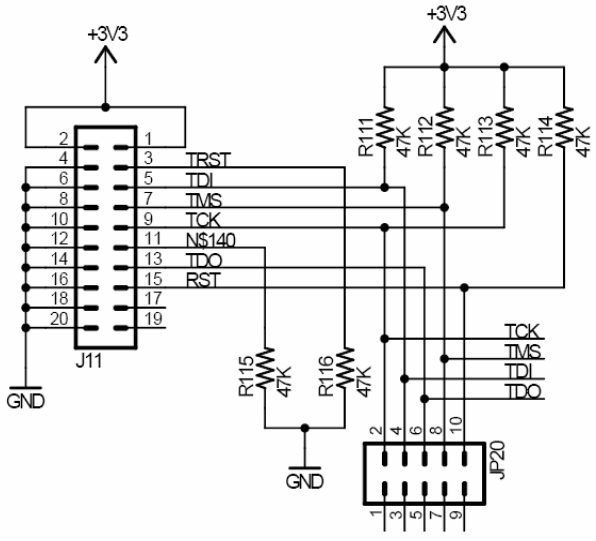

Figure 6: JTAG Interface Schematic

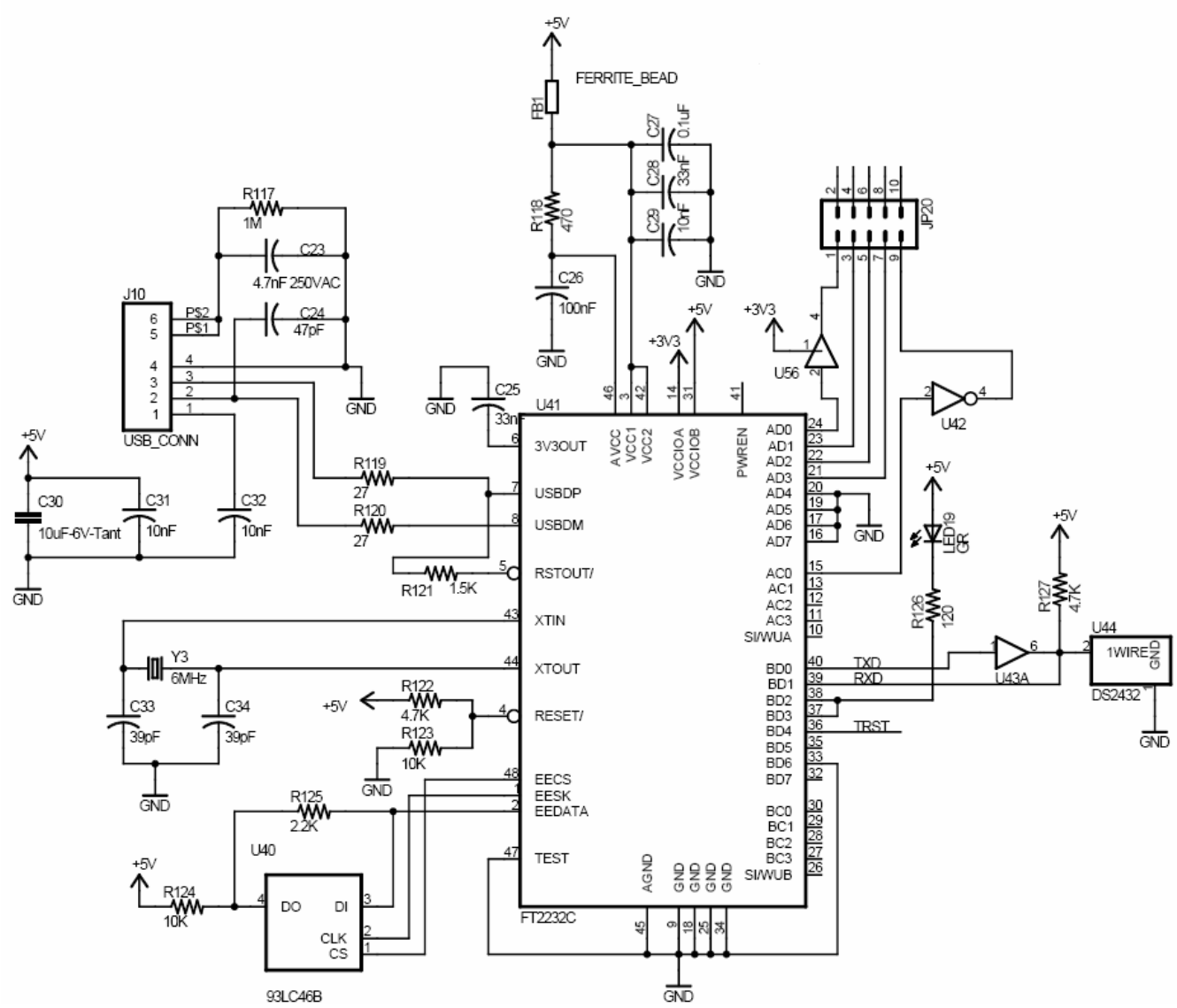

Figure 7: USB JTAG Interface Schematic

Customer Design

The fido1100's superior interface flexibility gives the designer a variety of choices in the design of their boards. The designer should consider what their application is calling for and slant their design accordingly. For example, if the board design calls for JTAG interfaces on every board, the designer might find it more cost effective to have on-board firmware connecting to a USB interface. Another scenario may require no debug interface once the board is fully tested. In this instance, only a limited JTAG interface would be required.

This application note in concert with the existing EDK development board provide representative examples on how to interface with a Macraigor wiggler JTAG interface (either external or on-board). The Interface Schematics (Figure 6 and Figure 7) are tested and verified methods of successful integration of the JTAG interface. Please feel free to use these examples as a guide for your JTAG implementation.

Note: An on-board wiggler requires licensing firmware from Macraigor.

Conclusions

This Application Note shows how simple it is to use the **fido1100** JTAG Interface.

Thank You

Thank you for taking the time to review this application note. We hope you have found the information included in this application useful and easy to understand. Please feel free to contact the Innovasic Support Team any time with questions or comments.

Innovasic Support Team
3737 Princeton NE
Suite 130
Albuquerque, NM, 87107

(505) 883-5263

support@innovasic.com
http://www.innovasic.com