

SISTEMI EMBEDDED

AA 2012/2013

JTAG CIRCUITRY

JTAG DEBUG MODULE

JTAG-UART PERIPHERAL

Joint Test Action Group (JTAG) (1)

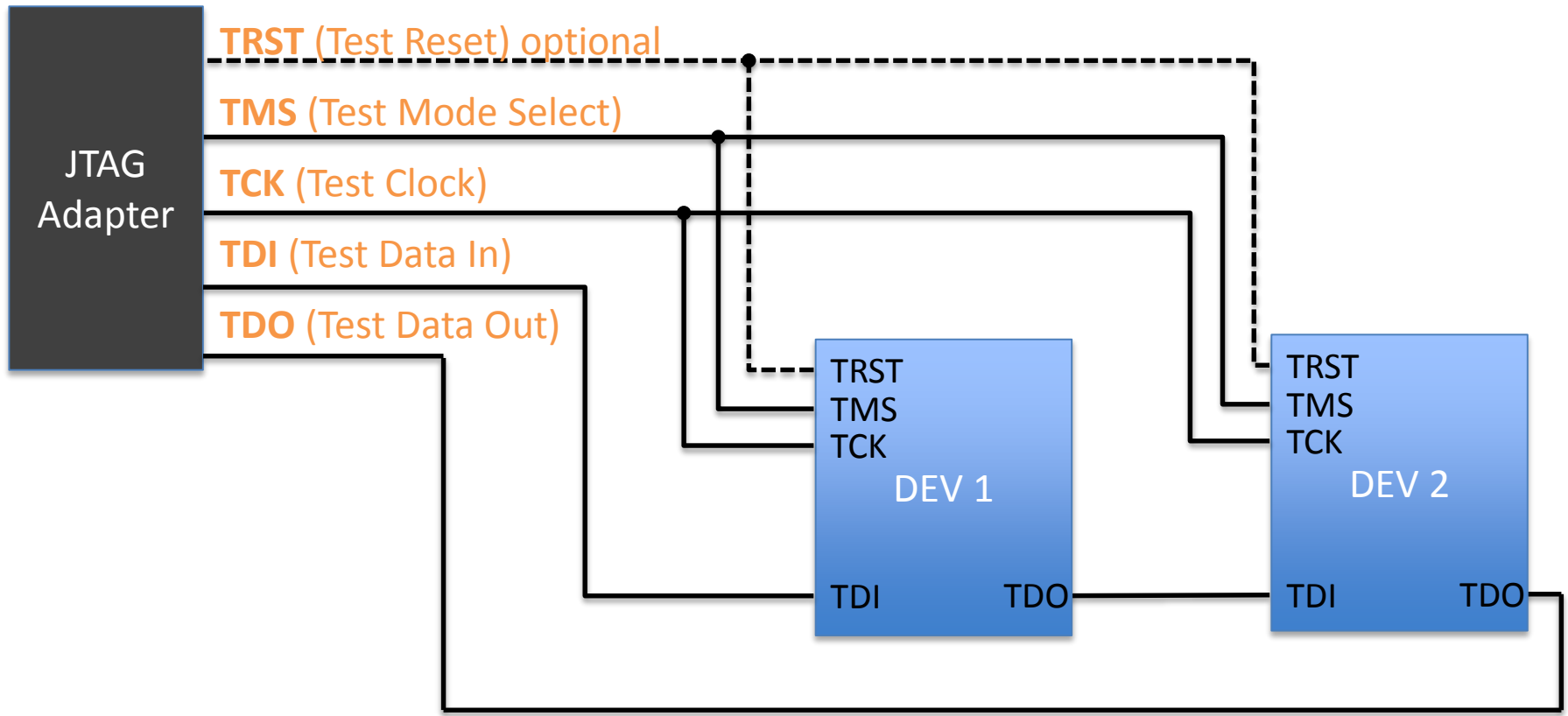
- Established in 1985 to develop a method to test populated PCBs
 - A way to access IC pins organized in a boundary-scan
- The method was standardized in 1990 as IEEE 1149.1 (**Standard Test Access Port and Boundary-Scan Architecture**)
 - A 1994 addendum defined the Boundary Scan Description Language (BSDL)

Joint Test Action Group (JTAG) (2)

- Today, JTAG is widely used also as communication means to access IC sub-blocks:
 - For debugging purposes in processors
 - For programming purposes in processors, CPLDs, FPGAs
 - ...
- JTAG acts as a communication interface to serially read/write internal registers

JTAG interface

Test Access Port (TAP)



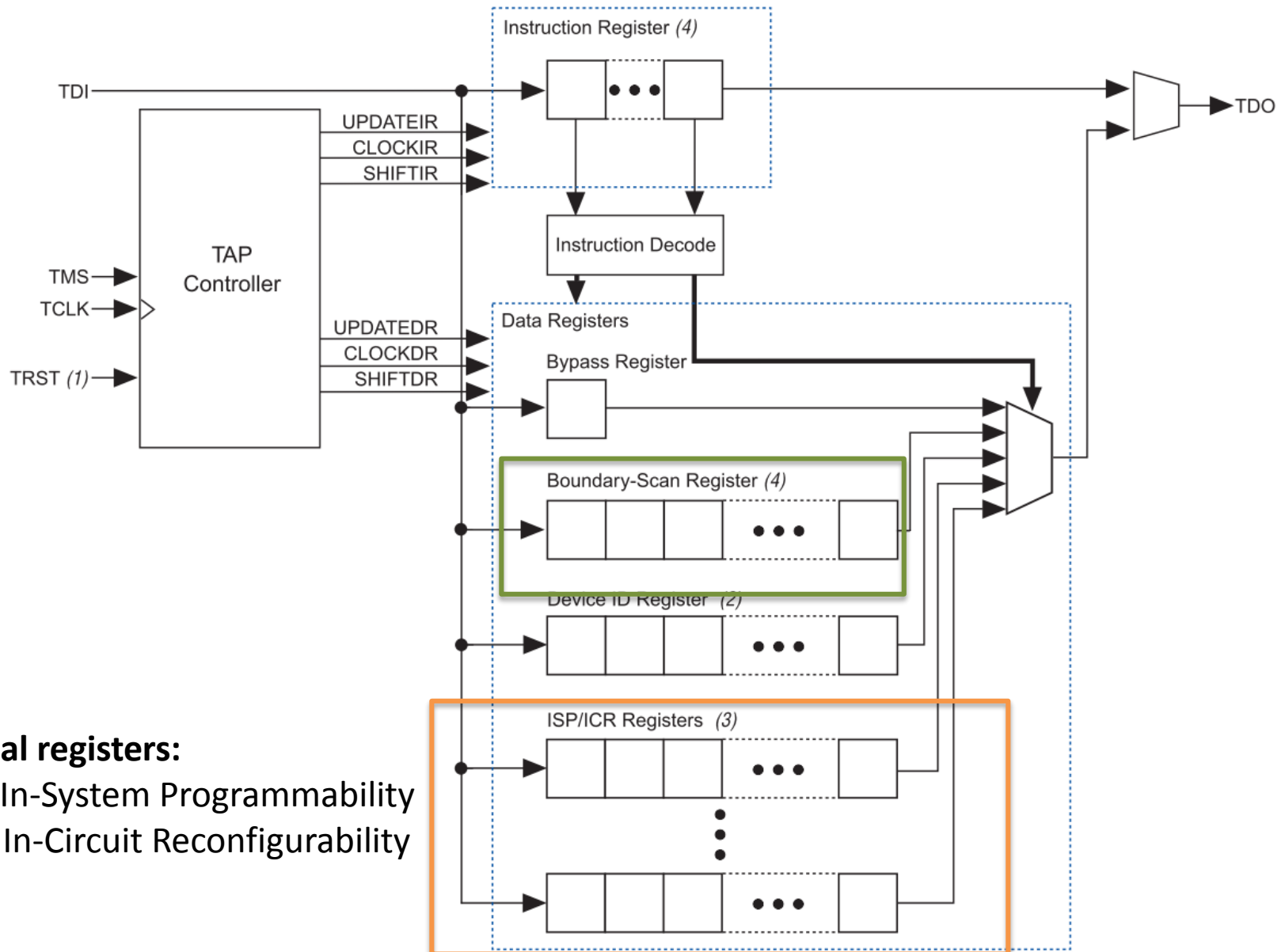
JTAG signals (1)

- **TDI (Test Data In):** Serial input pin for instructions as well as (test and programming) data. Bits are shifted in on the rising edge of TCK.
- **TDO (Test Data Out):** Serial data output pin for instructions as well as (test and programming) data. Bits are shifted out on the falling edge of TCK. This pin is tri-stated if bits are not being shifted out of the device.
- **TMS (Test Mode Select):** Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur at the rising edge of TCK (TMS is evaluated on the rising edge of TCK). Therefore, TMS must be set up before the rising edge of TCK.

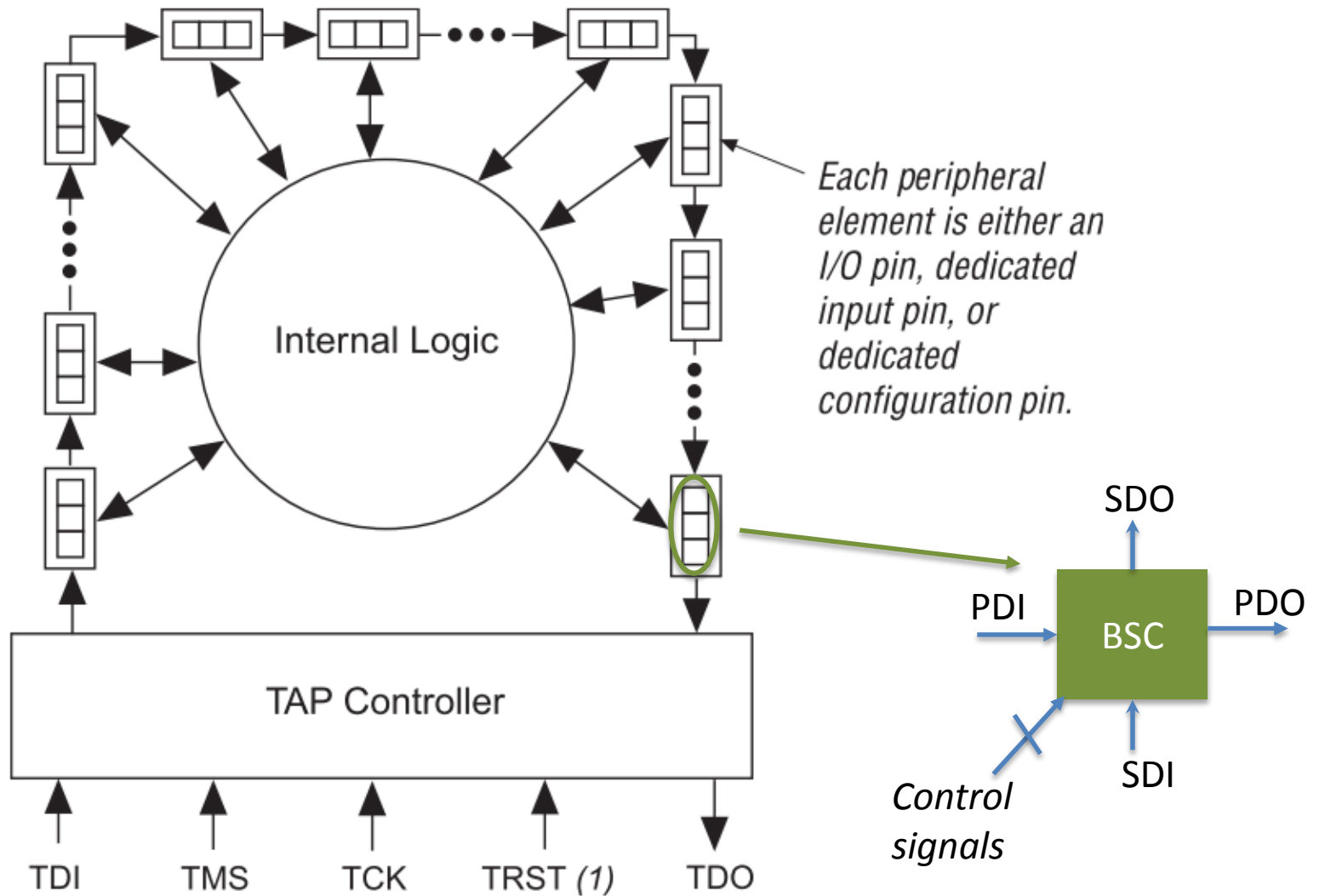
JTAG signals (2)

- **TCK (Test Clock input):** The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge.
- **TRST (Test Reset input):** Active-low input to asynchronously reset the boundary-scan circuit. (TRST is optional according to IEEE Std. 1149.1). This pin should be driven low when not in boundary scan operation and for non-JTAG users the pin should be permanently tied to GND.

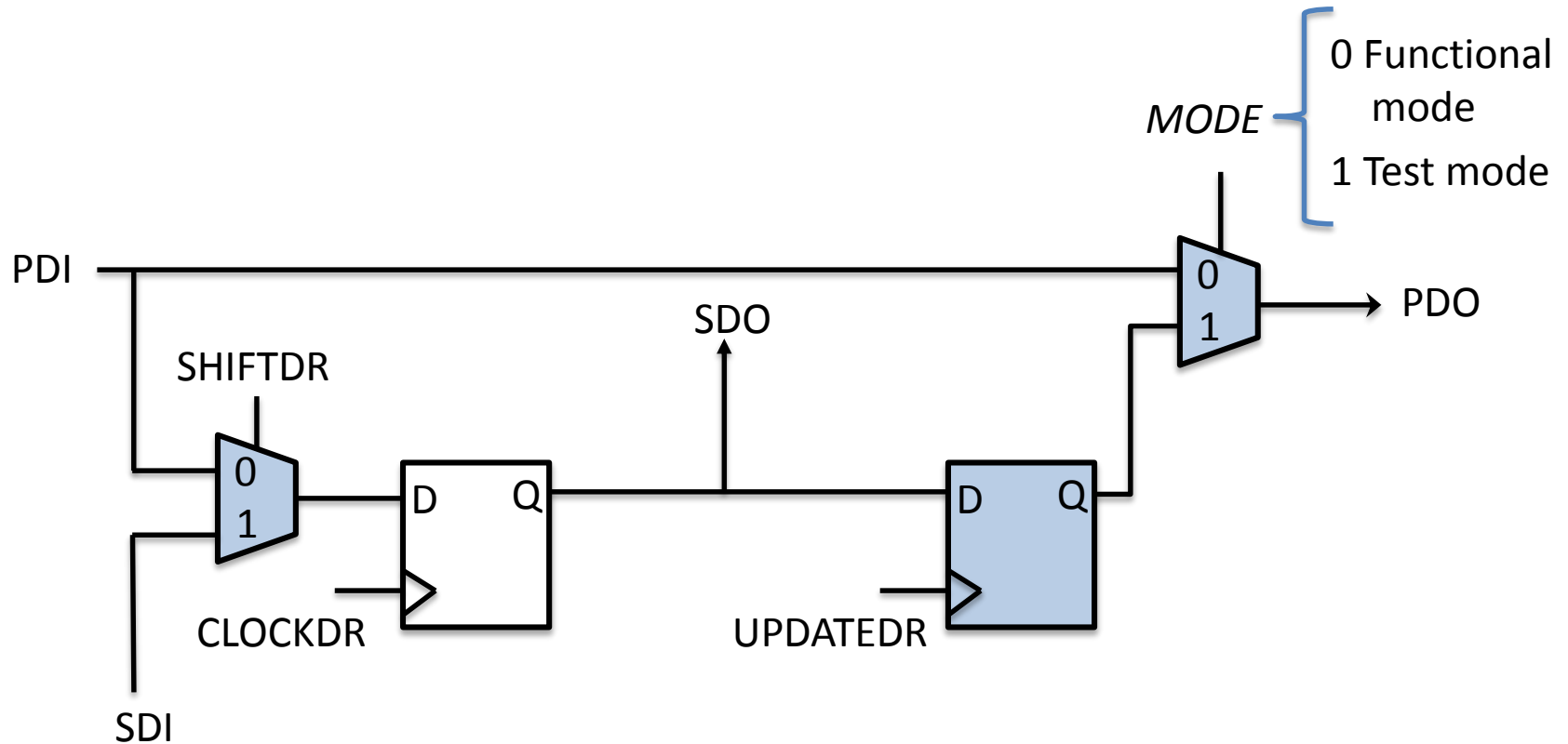
JTAG circuitry (in Altera devices)



Boundary Scan Cell (BSC) (1)



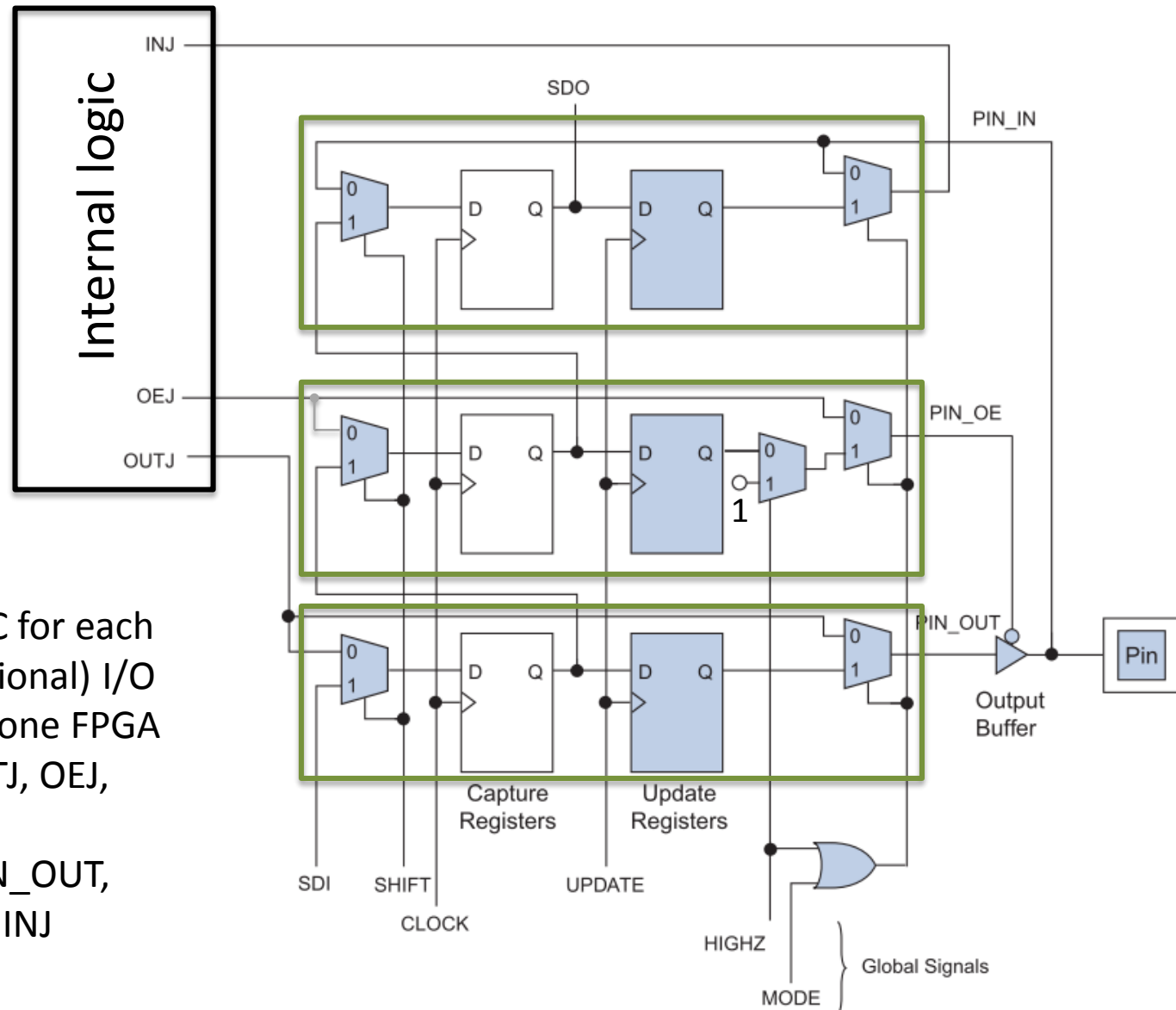
Boundary Scan Cell (BSC) (2)



Each BSC can:

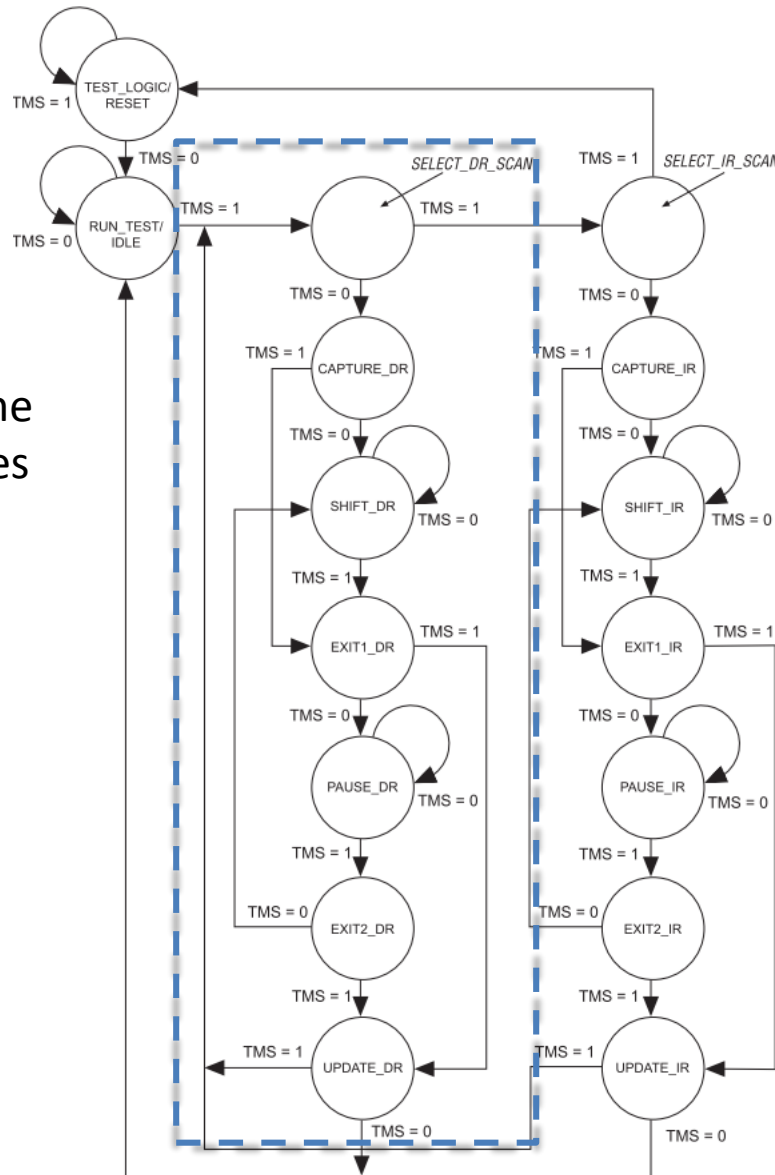
- Capture data on its parallel input PDI
- Update data on its parallel output PDO
- Serially shift SDI to SDO
- Behave transparently $PDO = PDI$

Boundary Scan Cell (BSC) (3)

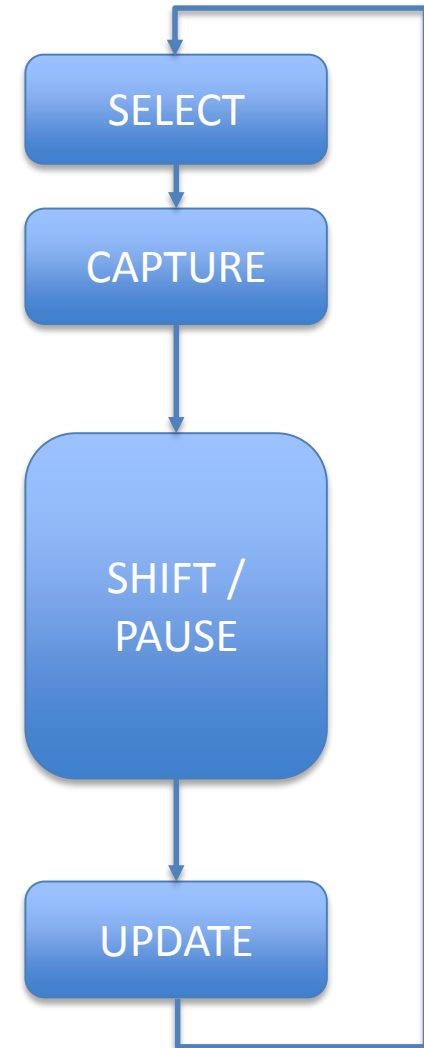


- 3-bit BSC for each (bidirectional) I/O of a Cyclone FPGA
- PDI: OUTJ, OEJ, PIN_IN
- PDO: PIN_OUT, PIN_OE, INJ

TAP controller



Finite State Machine (FSM) with 16 states

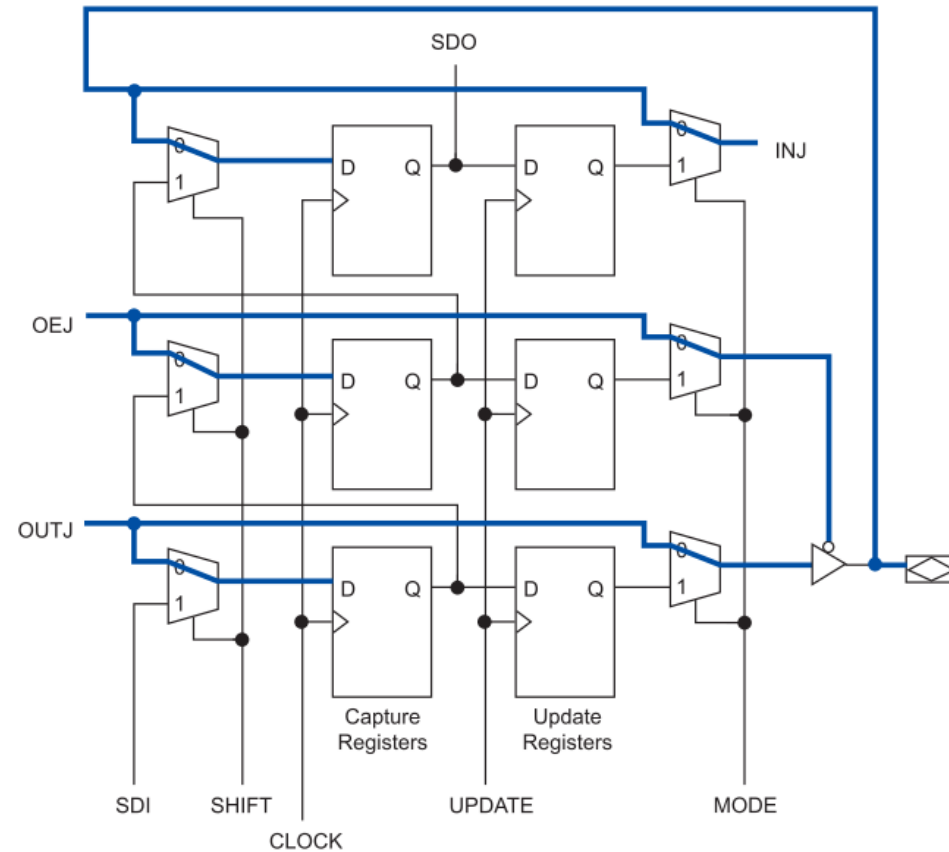


Standard instructions

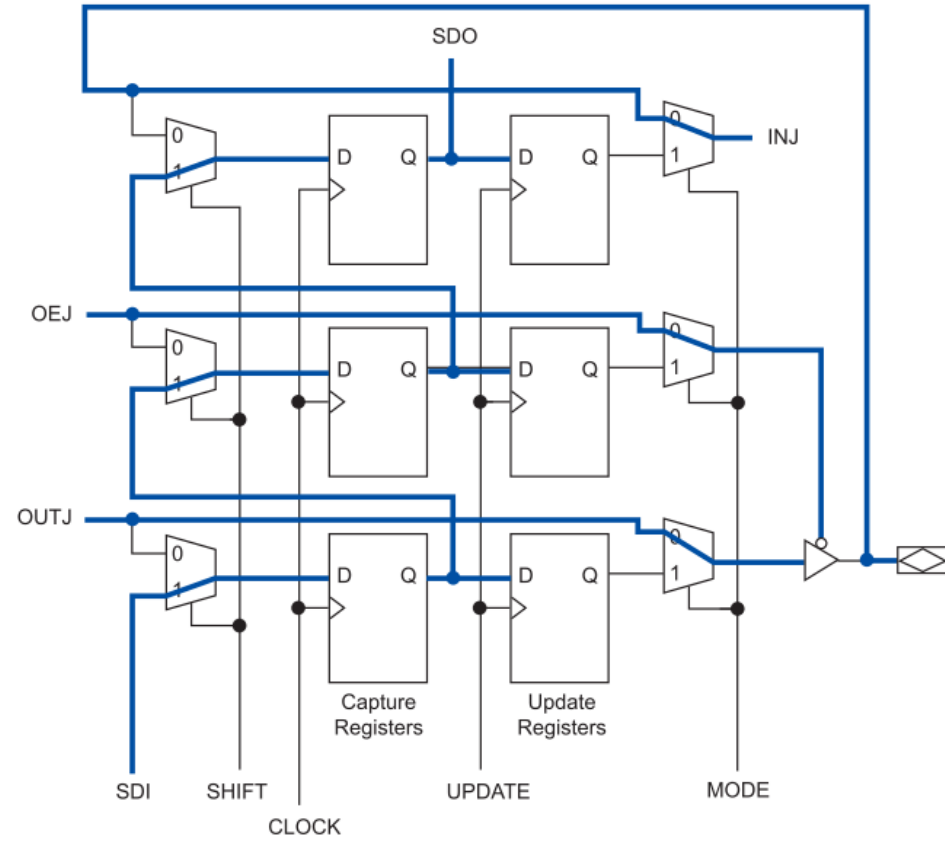
Instruction	Selected Data Register / (Mode)
Mandatory	
Extest	Boundary scan (Test mode)
Bypass	Bypass (1-bit bypass register between TDI and TDO)
Sample/Preload	Boundary scan (Functional mode)
Optional	
Intest	Boundary scan (Test mode)
Idcode	IDCODE register
Usercode	USERCODE register
Runbist	Result register
Clamp	Bypass (I/O hold to the state defined by the boundary scan register)
HighZ	Bypass (I/O in Hi-Z)

Sample/Preload instructions

Capture phase

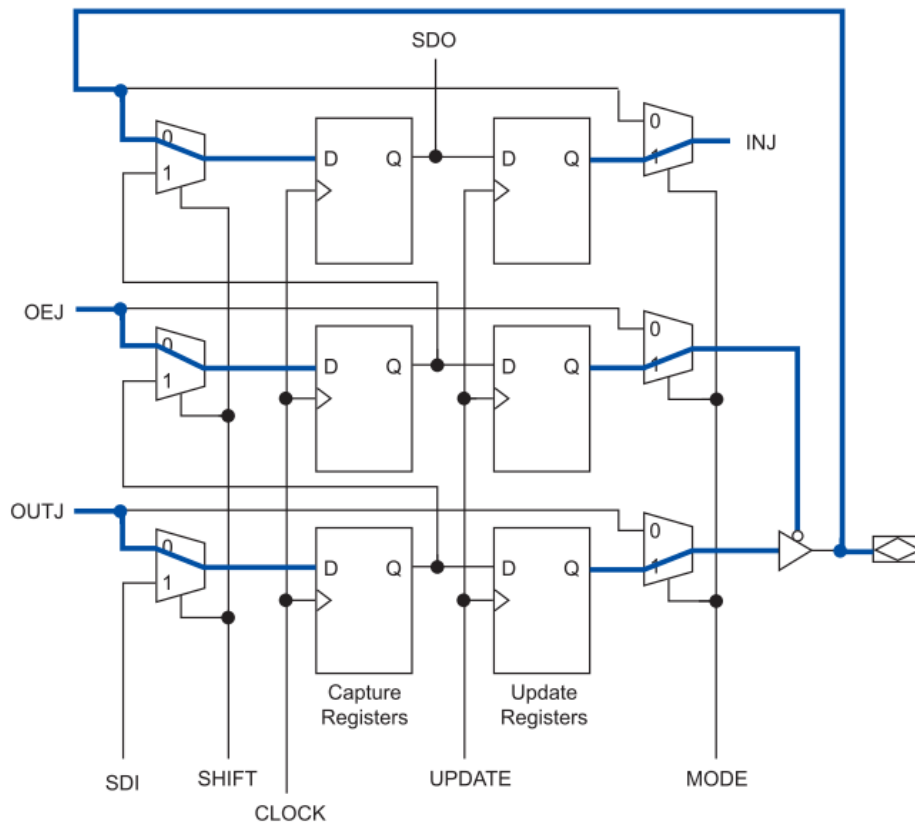


Shift and Update phases

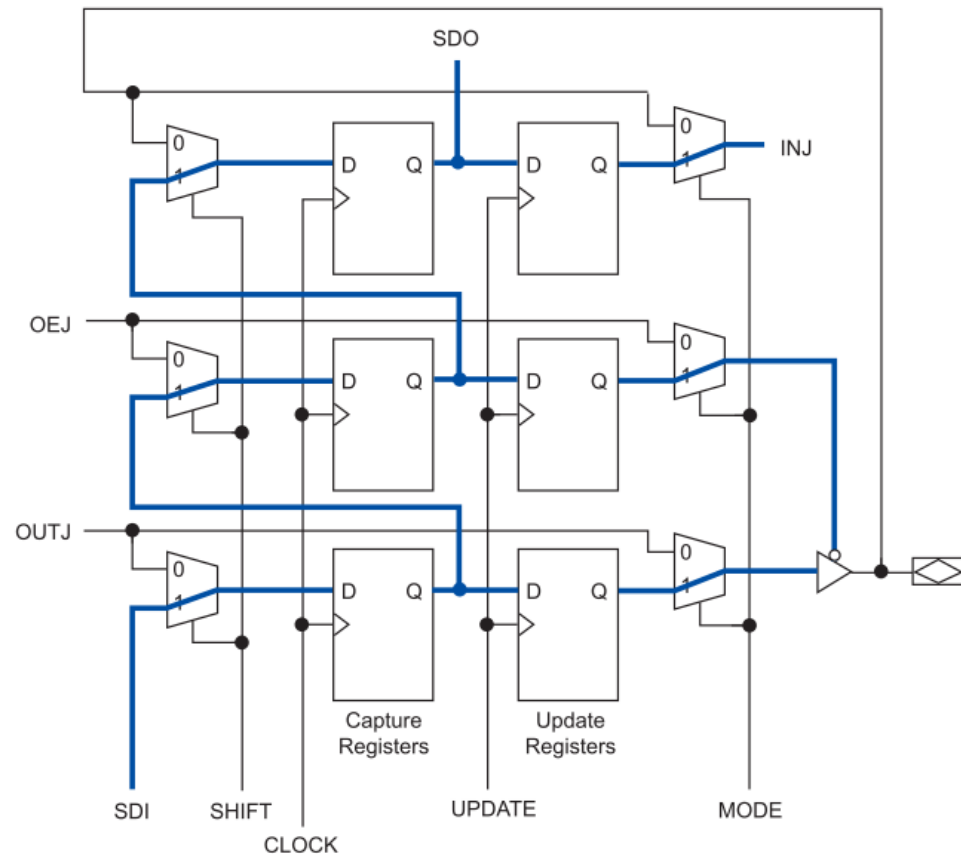


Exttest/Intest instruction

Capture phase



Shift and Update phases



JTAG Debug Module (1)

- Provides on-chip emulation to control the processor remotely from a host PC. Software debugging tools communicate with the JTAG debug module and make it possible to:
 - Download programs to memory
 - Start and stop execution
 - Set breakpoints and watchpoints
 - Analyze registers and memory
 - Collect real-time execution trace data

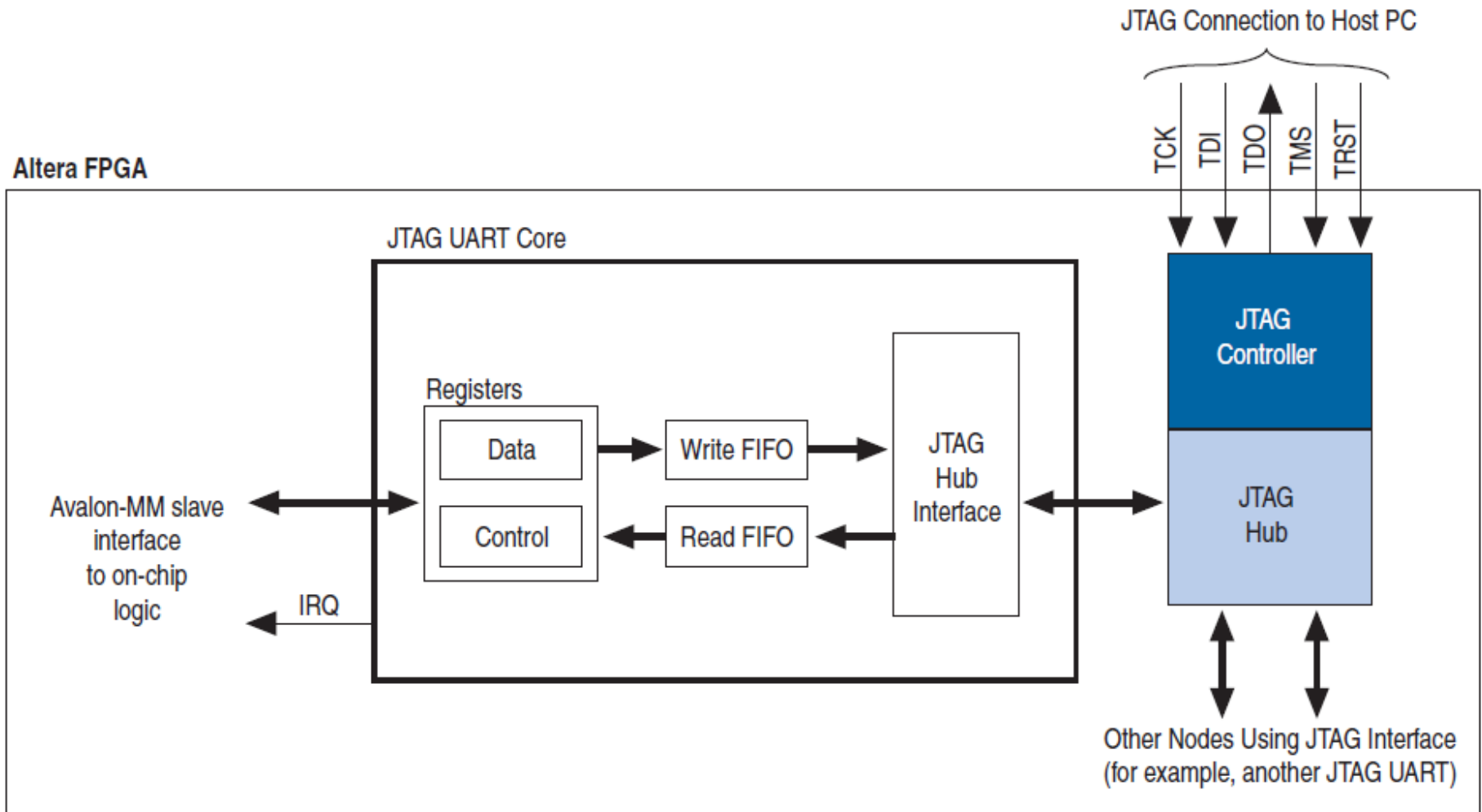
JTAG Debug Module (2)

- Piece of hardware (implemented with FPGA logic and memory resources) placed between the JTAG circuitry and the processor
- The JTAG debug module takes control of the processor by asserting a hardware break or inserting a break instruction in the program memory to be executed
 - A break causes the software routine located at the break address to be executed
 - The break address is chosen when the processor is generated

JTAG-UART Core

- Allows characters to be serially transmitted between a Nios processor and a host PC using the JTAG circuitry present in the FPGA and a download cable such as USB-Blaster
- Hides the complexity of the JTAG circuitry
- Is supported by HAL system library
 - Character-mode generic device model (Unix-like routines: *open*, *read*, *write*, ...)
 - C standard I/O functions (*printf*, *getchar*,...)

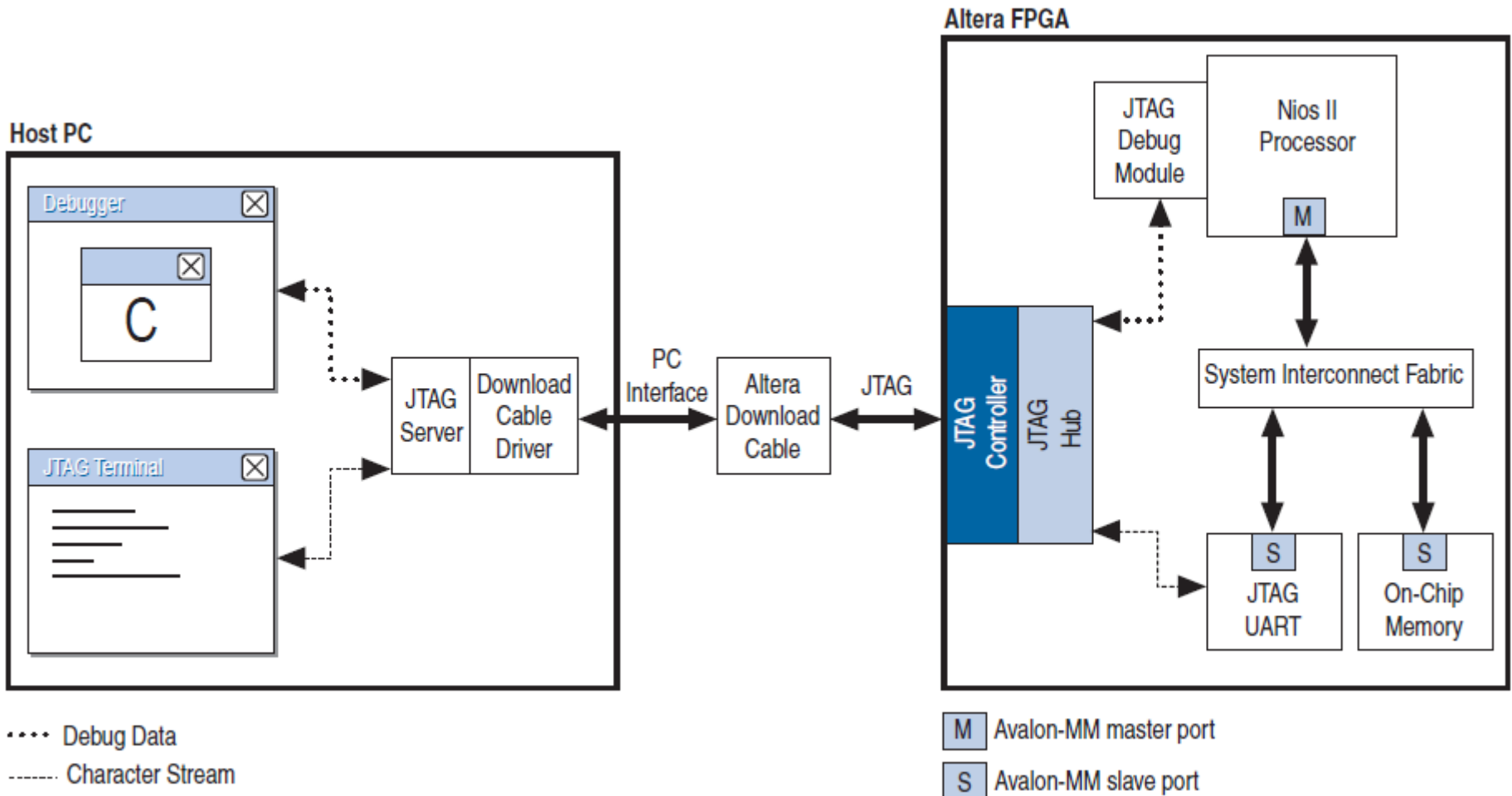
JTAG-UART Core block diagram



Built-In Feature of Altera FPGA

Automatically Generated by Quartus II Software

Host-Nios II processor connection



Hardware configuration (1)

- **Read/Write FIFO settings:**
 - Buffer depth = 2^N B, $N = 3 - 15$ (default: 64; $N=6$)
 - IRQ threshold (default: 8)
 - IRQ is asserted when the number of remaining bytes in the Read FIFO, which can still be written (filled) by the JTAG circuitry, equals the IRQ *read_threshold*
 - IRQ is asserted when the number of remaining bytes in the Write FIFO, which can still be read (emptied) by the JTAG circuitry, equals the IRQ *write_thresold*
 - Construct using registers instead of memory blocks
 - 1 B consumes roughly 11 logic elements (LEs)

Hardware configuration (2)

- **Simulation settings:**

- These settings control the generation of the JTAG UART core simulation model
 - Fixed input stream loaded in the Read FIFO at reset
 - Macros for the ModelSim simulator to generate Interactive Windows
 - To display the content of the Write FIFO
 - To also write the Read FIFO instead of the fixed input stream
- They do not affect the hardware generation

Register map (1)

- Data and Control registers

Offset	Register Name	R/W	Bit Description														
			31	...	16	15	14	...	11	10	9	8	7	...	2	1	0
0	data	RW	RAVAIL			RVALID	Reserved					DATA					
1	control	RW	WSPACE			Reserved				AC	WI	RI	Reserved			WE	RE

- Data register

Bit(s)	Name	Access	Description
[7:0]	DATA	R/W	The value to transfer to/from the JTAG core. When writing, the DATA field holds a character to be written to the write FIFO. When reading, the DATA field holds a character read from the read FIFO.
[15]	RVALID	R	Indicates whether the DATA field is valid. If RVALID=1, the DATA field is valid, otherwise DATA is undefined.
[32 :16]	RAVAIL	R	The number of characters remaining in the read FIFO (after the current read).

Register map (2)

- Control register

Bit(s)	Name	Access	Description
0	RE	R/W	Interrupt-enable bit for read interrupts.
1	WE	R/W	Interrupt-enable bit for write interrupts.
8	RI	R	Indicates that the read interrupt is pending.
9	WI	R	Indicates that the write interrupt is pending.
10	AC	R/C	Indicates that there has been JTAG activity since the bit was cleared. Writing 1 to AC clears it to 0.
32:16	WSPACE	R	The number of spaces available in the write FIFO.

31

- AC is set after an application on the host PC has polled the JTAG UART core via the JTAG interface
- Once set, the AC bit remains set until it is explicitly cleared via the Avalon interface. Writing 1 to AC clears it
- Embedded software can examine the AC bit to determine if a connection exists to a host PC

Interrupt behaviour

- JTAG can generate an interrupt when the Read FIFO is almost full (*read_threshold*) or the Write FIFO is almost empty (*write_threshold*)
 - The write interrupt is cleared by writing characters to fill the write FIFO beyond the *write_threshold*
 - The read interrupt is cleared by reading characters from the read FIFO
 - The read interrupt condition is also set if there is at least one character in the read FIFO and no more characters are expected

Software programming model

ANSI C
(Standard Library)

streams (using file pointer)
fopen(), fread(), fwrite(),
fclose(), fprintf()...

Eg. FILE *fp;
fp=fopen(...);
fread(...,fp);

HAL
(Unix-like functions)

HAL (using file descriptor)
open(), read(), write(),
close(), ioctl(), ...

Eg. int fd;
fd = open(...);
read(fd,...);

**Device
Driver**



altera_avalon_jtag_uart.c



altera_avalon_jtag_uart.h



altera_avalon_jtag_uart_regs.h

system.h

JTAG UART
Core

DATA REGISTER

CONTROL REGISTER

File descriptor/pointer

- File descriptors (integer) are used by low-level I/O functions: `open`, `read`, `write`, `close`,...
- File pointers manage streams using high-level functions (buffered operations):
 - Unformatted I/O: *fgetc*, *fputc*
 - Formatted I/O: *fprintf*, *fscanf*,...
 - Eg. *stdin*, *stdout*, *stderr* are specific streams which do not require open and close operations and for which are defined *printf*, *getchar*, *scanf*,... functions are defined

Blocking and Non blocking (1)

- A **blocking read** waits until the required bytes are available
- To allow the read function to immediately return if no data are available the file descriptor must be opened with NONBLOCK flag
 - `fd = open("/dev/<your device name>",
O_NONBLOCK | O_RDWR);`

Blocking and Non blocking (2)

- When using a file pointer, we need to retrieve the underlying file descriptor

```
int fd = fileno(fp);  
int flags = fcntl(fd, F_GETFL);  
fcntl(fd, F_SETFL, flags | O_NONBLOCK);
```

- These header file must be included:
 #include <unistd.h>
 #include <fcntl.h>
- When no data are available to read, EOF is returned by the read function

Putting into practice (1)

- Write a program that displays the ASCII code of a character received from the JTAG UART
 - *stdin*, *stdout*, *stderr* streams must be disconnected from JTAG UART device (BSP Editor)
 - Note that *printf* is no longer available!
 - Use a **file pointer** to access the JTAG UART device
 - *fgetc* reads a character from the stream
 - *fprintf* sends formatted strings to the Host PC
 - Use nios2-terminal to connect to the processor
 - The Nios II console on Eclipse must be turned off!

Putting into practice (2)

- Write a program that displays the ASCII code of a character received from the JTAG UART
 - See the effect of blocking read; for example inserting some other operations in the main loop, such as reading the DE2 slider switches and updating the Red LEDs status accordingly
 - Try to use the non-blocking flag

References

- Altera “IEEE 1149.1 JTAG Boundary-Scan Testing,” AN39 ver. 6.0, June 2005
- Altera, “Nios II Processor Reference Handbook,” [*n2cpu_nii5v1.pdf*](#)
 - Processor Architecture – JTAG Debug Module
- Altera “Embedded Peripherals User Guide,” [*ug_embedded_ip.pdf*](#)
 - 6. JTAG UART Core
- Altera “Nios II Software Developer’s Handbook,” [*n2sw_nii5v2.pdf*](#)
 - Chapter 6. Developing Programs Using the Hardware Abstraction Layer
 - Chapter 14. HAL API Reference