# Example: how to get the temperature of one COSTAR chip

Temperature = costar[ladder\_nb][module\_nb].adc1[3]

## Function call graph



## jtagCostar.c

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void get\_adc1\_one\_costar(unsigned char data[4],unsigned short costar\_nb,chip \*map)**

**Description: this function gets the adc1 register of the costar "costar\_nb" and puts the values in the array "data". the conversion into floating point value is done according to the type of data (temperature, voltage supply, etc..)**

**Note: data[0] : external channel (V) data[1] : GND – VSS (V)**

**data[2] : VDD – VSS (V) data[3] : temperature (°C)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

void get\_adc1\_one\_costar**(**unsigned char data**[**4**],**unsigned short costar\_nb**,**chip **\***map**)**

**{**

unsigned long list\_in**;**

unsigned short nb\_costar**,**chan**;**

**if** **((**nb\_costar **=** map\_costar\_nb**(**map**))==**0**)**

**return** **;**

list\_in **=** get\_one\_costar**(**CO\_ADC\_1**,**CO\_ADC\_SZ**,**costar\_nb**,**map**);**

**for** **(**chan**=**0**;**chan**<**4**;**chan**++) {**

data**[**chan**]=((**unsigned char**)** **(**list\_in**>>(**chan**\***CO\_ADC\_SZ**/**4**))&**255**);**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**unsigned long get\_one\_costar(unsigned long instruction, unsigned short instr\_siz, unsigned short costar\_nb, chip \*map)**

**Description: this function send the instruction to the costar "costar\_nb" (map descibes the chain), then get the data from this costar.**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long get\_one\_costar**(**unsigned long instruction**,** unsigned short instr\_siz**,** unsigned short costar\_nb**,**chip **\***map**)**

**{**

unsigned short stream\_in**[**MAX\_LENGTH\_STREAM**],**stream**[**MAX\_LENGTH\_STREAM**];**

unsigned short nb\_chip**,**i**,**icost**,**cost\_i**=**0**;**

**if** **(**map\_costar\_nb**(**map**)==**0**)** **return** 0**;**

nb\_chip **=** map\_nb**(**map**);**

**for** **(**i**=**0**,**icost**=**0**;**i**<**nb\_chip**;**i**++)** /\* find the costar in the chain \*/

**if** **(**map**[**i**]==**COSTAR**) {**

**if** **(**icost**==**costar\_nb**)** **{**

cost\_i**=**i**;**

**break;**

**}**

**else** icost**++;**

**}**

init\_stream**(**stream**,**CO\_IR\_SZ**\***nb\_chip**);** /\* fill the output stream with BYPASS instructions \*/

set\_stream**(**stream**,**cost\_i**\***CO\_IR\_SZ**,**instruction**,**CO\_IR\_SZ**);** /\* put the instruction for the COSTAR in the output stream. \*/

scan\_ir**(**stream**,**CO\_IR\_SZ**\***nb\_chip**,**stream\_in**);** /\* send the instruction stream \*/

circulate\_dr**(**nb\_chip**\***CO\_BYPASS\_SZ**+**instr\_siz**-**1**,**stream**);** /\* get the data stream \*/

**return** get\_stream**(**stream**,**cost\_i**\***CO\_BYPASS\_SZ**,**instr\_siz**);** /\* return the data \*/

**}**

## jtagCommon.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: scan\_ir()

Summary: Scans a bit stream into the TAP instruction register

Usage: void scan\_ir(output, length, input)

unsigned short \*output;

unsigned short length;

unsigned short \*input;

Description: This routine scans an arbitrary length bit stream into

the target TAP controller instruction register. When a

scan starts, the RX-FIFO holds garbage, so we always shift

out a 16-bit dummy value first to take out the garbage.

This function has three parameters. The 'output' variable

is an address to an array of 16-bit values to be scanned

out the JTAG port. The 'length' parameter specifies the

length of the bitstream. The 'input' variable is an address

to an array of 16-bit values where the data scanned in is

to be stored.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void scan\_ir**(**unsigned short **\***output**,**unsigned short length**,**unsigned short **\***input**)**

**{**

unsigned short x**,**numregs**;**

/\* determine number of registers to use in data transfer \*/

numregs **=** length **/** 16**;**

**if ((**numregs **\*** 16**)** **!=** length**)**

**++**numregs**;**

write\_tbc**(**CONTROL3**,**0x008f**);** /\* start TCK \*/

write\_tbc**(**CONTROL4**,**0x0280**+**tck\_delay**);**/\* connect TDI0 to data-in-src \*/

/\* send recirculate data from shifter FIFO \*/

/\* ignore received data \*/

/\* FIFO length = 16 \*/

write\_tbc**(**MAJOR**,**0x300f**);** /\* SCAN command, use counter1 \*/

/\* scan IR, end in pause-ir \*/

write\_tbc**(**CNT1UP0**,**15**);** /\* counter1 update value = 16 bits \*/

write\_tbc**(**MINOR**,**0x6001**);** /\* OPERATE2, update counter1 \*/

write\_tbc**(**MINOR**,**0x4080**);** /\* OPERATE0, begin SCAN command \*/

wait\_jtag\_state**(**PAUSE\_IR**);** /\* wait until SCAN command finishes \*/

/\* and JTAG bus goes to pause-ir \*/

write\_tbc**(**CONTROL4**,**0x0780**+**tck\_delay**);** /\* connect TDI0 to data-in-src \*/

/\* send overwrite data from write buffer \*/

/\* read receive data into read buffer \*/

/\* FIFO length = 16 \*/

write\_tbc**(**MAJOR**,**0x300a**);** /\* SCAN command, use counter1 \*/

/\* scan IR, end in run-test-idle \*/

write\_tbc**(**CNT1UP0**,**length **-** 1**);** /\* counter1 update value = length - 1 \*/

write\_tbc**(**MINOR**,**0x6001**);** /\* OPERATE2, update counter1 \*/

write\_tbc**(**MINOR**,**0x4080**);** /\* OPERATE1, begin SCAN command \*/

**for (**x **=** 0 **;** x **<** numregs **;** **++**x**)** **{**

write\_buf**(**output**[**x**]);** /\* scan out data \*/

read\_buf**(&**input**[**x**]);** /\* read data from target \*/

**}**

wait\_jtag\_state**(**RUN\_TEST\_IDLE**);** /\* wait until SCAN command finishes \*/

/\* and JTAG bus goes to run-test-idle \*/

write\_tbc**(**CONTROL3**,**0x208f**);** /\* stop TCK \*/

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: scan\_dr()

Summary: Scans a bit stream out the TAP data path

Usage: void scan\_dr(output, length, input)

unsigned short \*output;

unsigned short length;

unsigned short \*input;

Description: This routine scans an arbitrary length bit stream out

the target TAP controller data path. When a scan starts,

the RX-FIFO holds garbage, so we always shift out a

16-bit dummy value first to take out the garbage.

This function has three parameters. The 'output' variable

is an address to an array of 16-bit values to be scanned

out the JTAG port. The 'length' parameter specifies the

length of the bitstream. The 'input' variable is an address

to an array of 16-bit values where the data scanned in is

to be stored.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void scan\_dr**(**unsigned short **\***output**,**unsigned short length**,**unsigned short **\***input**)**

**{**

unsigned short x**,**numregs**;**

/\* this routine scans data thru the JTAG data path \*/

/\* determine number of registers to use in data transfer \*/

numregs **=** length **/** 16**;**

**if ((**numregs **\*** 16**)** **!=** length**)**

**++**numregs**;**

write\_tbc**(**CONTROL3**,**0x008f**);** /\* start TCK \*/

write\_tbc**(**CONTROL4**,**0x0280**+**tck\_delay**);** /\* connect TDI0 to data-in-src \*/

/\* send recirculate data from shifter FIFO \*/

/\* ignore received data \*/

/\* FIFO length = 16 \*/

write\_tbc**(**MAJOR**,**0x3006**);** /\* SCAN command, use counter1 \*/

/\* scan DR, end in pause-dr \*/

write\_tbc**(**CNT1UP0**,**15**);** /\* counter1 update value = 16 bits \*/

write\_tbc**(**MINOR**,**0x6001**);** /\* OPERATE2, update counter1 \*/

write\_tbc**(**MINOR**,**0x4080**);** /\* OPERATE0, begin SCAN command \*/

wait\_jtag\_state**(**PAUSE\_DR**);** /\* wait until SCAN command finishes \*/

/\* and JTAG bus goes to pause-dr \*/

write\_tbc**(**CONTROL4**,**0x0780**+**tck\_delay**);** /\* connect TDI0 to data-in-src \*/

/\* send overwrite data from write buffer \*/

/\* read receive data into read buffer \*/

/\* FIFO length = 16 \*/

write\_tbc**(**MAJOR**,**0x3002**);** /\* SCAN command, use counter1 \*/

/\* scan DR, end in run-test-idle \*/

write\_tbc**(**CNT1UP0**,**length **-** 1**);** /\* counter1 update value = length - 1\*/

write\_tbc**(**MINOR**,**0x6001**);** /\* OPERATE2, update counter1 \*/

write\_tbc**(**MINOR**,**0x4080**);** /\* OPERATE1, begin SCAN command \*/

**for (**x **=** 0 **;** x **<** numregs **;** **++**x**)** **{**

write\_buf**(**output**[**x**]);** /\* scan out data \*/

read\_buf**(&**input**[**x**]);** /\* read data from target \*/

**}**

wait\_jtag\_state**(**RUN\_TEST\_IDLE**);** /\* wait until SCAN command finishes \*/

/\* and JTAG bus goes to run-test-idle \*/

write\_tbc**(**CONTROL3**,**0x208f**);** /\* stop TCK \*/

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: circulate\_dr()

Summary: Circulates a bit stream thru the TAP controller data path

Usage: void circulate\_dr(length, data)

unsigned short length;

unsigned short \*data;

Description: This routine circulates an arbitrary length bit stream thru

the target TAP controller data path. Data doesnt actually

start circulating until the first 32 bits of data are sent

out. The first 16 bits are sent to get valid data into the

RX-FIFO, and the second 16 bits force the first valid data

out of the scan path. This function has two parameters.

The 'length' parameter specifies how many bits to

circulate. The 'data' parameter is an address to an array

of 16-bit values where the data scanned in is to be stored.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void circulate\_dr**(**unsigned short length**,**unsigned short **\***output**)**

**{**

unsigned short x**,**numregs**;**

/\* determine number of registers to use in data transfer \*/

numregs **=** length **/** 16**;**

**if ((**numregs **\*** 16**)** **!=** length**)**

**++**numregs**;**

write\_tbc**(**CONTROL3**,**0x008f**);** /\* start TCK \*/

write\_tbc**(**CONTROL4**,**0x0280**+**tck\_delay**);** /\* connect TDI0 to data-in-src \*/

/\* send recirculate data from shifter FIFO \*/

/\* ignore receive data \*/

/\* FIFO length = 16 \*/

write\_tbc**(**MAJOR**,**0x3006**);** /\* SCAN, use counter1 \*/

/\* scan DR, end in pause-dr \*/

write\_tbc**(**CNT1UP0**,**15**);** /\* counter1 update value = 16 bits \*/

write\_tbc**(**MINOR**,**0x6001**);** /\* OPERATE2, update counter1 \*/

write\_tbc**(**MINOR**,**0x4080**);** /\* OPERATE0, begin SCAN command \*/

wait\_jtag\_state**(**PAUSE\_DR**);** /\* wait until SCAN command finishes \*/

/\* and JTAG bus goes to pause-dr \*/

write\_tbc**(**CONTROL4**,**0x0780**+**tck\_delay**);** /\* connect TDI0 to data-in-src \*/

/\* send overwrite data from write buffer \*/

/\* read receive data into read buffer \*/

/\* FIFO length = 16 \*/

write\_tbc**(**MAJOR**,**0x3002**);** /\* SCAN command, use counter1 \*/

/\* scan DR, end in run-test-idle \*/

write\_tbc**(**CNT1UP0**,**length**-**1**+**16**);** /\* counter1 update value=length-1+16 \*/

write\_tbc**(**MINOR**,**0x6001**);** /\* OPERATE2, update counter1 \*/

write\_buf**(**0**);** /\* scan dummy value to get first data from target for recirculation \*/

write\_tbc**(**MINOR**,**0x4080**);** /\* OPERATE0, begin SCAN command \*/

/\* read and recirculate data into the target \*/

**for (**x **=** 0 **;** x **<** numregs **;** **++**x**)** **{**

read\_buf**(&**output**[**x**]);** /\* read result of DR-scan \*/

write\_buf**(**output**[**x**]);** /\* rescan data into target \*/

**}**

read\_buf**(&**x**);** /\* read last word which is garbage \*/

wait\_jtag\_state**(**RUN\_TEST\_IDLE**);** /\* wait until SCAN command finishes \*/

/\* and JTAG bus goes to run-test-idle \*/

write\_tbc**(**CONTROL3**,**0x208f**);** /\* stop TCK \*/

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

init\_stream(stream,length)

unsigned short \*stream

unsigned short length

return: nothing

Description: this function sets the stream to be all ones

(all ones instruction = BYPASS instruction, according to IEEE 1149.1)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init\_stream**(**unsigned short **\***stream**,**unsigned short length**)**

**{**

int i**,**n**;**

n**=**bit\_to\_word**(**length**);**

**for** **(**i**=**0**;**i**<(**n**+**1**);**i**++)**

stream**[**i**]=**0xffff**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

set\_stream(stream,loc\_bit,data,data\_sz)

unsigned short \*stream

unsigned short loc\_bit

unsigned long data

unsigned short data\_sz

return: nothing

Description: this function putsthe data in the stream at the location

corresponding to "loc\_bit".

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void set\_stream**(**unsigned short **\***stream**,**unsigned short loc\_bit**,**unsigned long data**,**unsigned short data\_sz**)**

**{**

unsigned short w1**,**w2**,**b1**,**b2**,**data\_mask**,**i**;**

w1 **=** loc\_bit**/**16**;** /\* nb of word where data begins \*/

b1 **=** loc\_bit **-** w1**\***16**;** /\* nb of bit where data begins in w1 \*/

w2 **=** bit\_to\_word**(**loc\_bit**+**data\_sz**);** /\* nb of word where data ends \*/

b2 **=** **(**loc\_bit**+**data\_sz**)** **-** w2**\***16**;** /\* nb of bit where data ends in w2 \*/

**if** **(**w1**==**w2**) {** /\* if the data fits in a single word of the stream \*/

**for** **(**data\_mask**=**0**,**i**=**0**;**i**<**16**;**i**++)**

**if** **((**i**<**b1**)||(**i**>=**b2**))** data\_mask**+=**1**<<**i**;**

stream**[**w1**]=(**stream**[**w1**]&**data\_mask**)+(**data**<<**b1**);** /\* fill this single word\*/

**return;**

**}**

**else** **{**

**for** **(**data\_mask**=**0**,**i**=**0**;**i**<**16**;**i**++)**

**if** **(**i**<**b1**)** data\_mask**+=**1**<<**i**;**

stream**[**w1**]=(**stream**[**w1**]&**data\_mask**)+((**data**<<**b1**)&**0xffff**);** /\* fill the first word \*/

**for** **(**data\_mask**=**0**,**i**=**0**;**i**<**16**;**i**++)**

**if** **(**i**>=**b2**)** data\_mask**+=**1**<<**i**;**

stream**[**w2**]=(**stream**[**w2**]&**data\_mask**)+(**data**>>(**data\_sz**-**b2**));** /\* fill the last word \*/

**if** **((**w1**+**1**)<**w2**)** /\* if the data fits in three words of the stream \*/

stream**[**w1**+**1**]=(**data**>>(**16**-**b1**))&**0xffff**;** /\* fill the middle word \*/

**}**

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

get\_stream(stream,loc\_bit,data\_sz)

unsigned short \*stream

unsigned short loc\_bit

unsigned short data\_sz

return: unsigned long data

Description: this function gets the data from the stream at the location

corresponding to "loc\_bit".

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long get\_stream**(**unsigned short **\***stream**,**unsigned short loc\_bit**,**unsigned short data\_sz**)**

**{**

unsigned short w1**,**w2**,**b1**,**b2**,**data\_mask**,**i**;**

unsigned long data**;**

w1 **=** loc\_bit**/**16**;** /\* nb of word where data begins \*/

b1 **=** loc\_bit **-** w1**\***16**;** /\* nb of bit where data begins in w1 \*/

w2 **=** bit\_to\_word**(**loc\_bit**+**data\_sz**);** /\* nb of word where data ends \*/

b2 **=** **(**loc\_bit**+**data\_sz**)** **-** w2**\***16**;** /\* nb of bit where data ends in w2 \*/

**if** **(**w1**==**w2**)** { /\* if the data fits in a single word of the stream \*/

**for** **(**data\_mask**=**0**,**i**=**0**;**i**<**16**;**i**++)**

**if** **((**i**>=**b1**)&&(**i**<**b2**))** data\_mask**+=**1**<<**i**;**

data**=(**stream**[**w1**]&**data\_mask**)>>**b1**;** /\* get the data \*/

**return** data**;**

**}**

**else** **{**

**for** **(**data\_mask**=**0**,**i**=**0**;**i**<**16**;**i**++)**

**if** **(**i**>=**b1**)** data\_mask**+=**1**<<**i**;**

data**=(**stream**[**w1**]&**data\_mask**)>>**b1**;** /\* get the part of the data from the first word \*/

**for** **(**data\_mask**=**0**,**i**=**0**;**i**<**16**;**i**++)**

**if** **(**i**<**b2**)** data\_mask**+=**1**<<**i**;**

data**+=((**unsigned long**)(**stream**[**w2**]&**data\_mask**))<<((**w2**-**w1**)\***16**-**b1**);** /\* get the part of the data from the last word \*/

**if** **((**w1**+**1**)<**w2**) {** /\* if the data fits in three words of the stream \*/

data**+=((**unsigned long**)**stream**[**w1**+**1**])<<(**16**-**b1**);** /\* get the part of the data from the middle word \*/

**}**

**}**

**return** data**;**

**}**