TMS320C6416 Power-On Self Test

David Abensur
Sebastien Tomas

Wireless Infrastructure Applications

ABSTRACT

The Power-On Self Test (POST) is designed to verify the operation of the TMS320C6416. Six modules are included in this test: Chk6xTest, MemoryEdmaTest, VcpTest, TcpTest, McbspTest, and TimerTest. These modules check the proper operation of the CPU core, internal memory and several on-chip peripherals (EDMA, McBSPs, timers, Viterbi and turbo decoder coprocessors).

It is important to note that this program only provides a confidence check. It is not as comprehensive as the tests done at production, which thoroughly check the device's logic, performance, and electrical parameters. The sample code described in this application report can be downloaded from http://www.ti.com/lit/zip/SPRA838.

Contents

List of Figures

List of Tables

1 Introduction

The Power-On Self Test (POST) is designed to verify the operation of the TMS320C6416. Six modules are included in this test: `Chk6xTest, MemoryEdmaTest, VcpTest, TcpTest, McbspTest` and `TimerTest`. These modules check the proper operation of the CPU core, internal memory and some on-chip peripherals (EDMA, McBSPs, Timers, Viterbi and turbo coprocessors). The test needs to be performed when the TMS320C6416 DSP is in its initial state (after reset). All modules consist of C-callable functions.

The CPU core module (Chk6xTest) is based on the *TMS320C62x Self-Check Test* (SPRA635). To be able to test the TMS320C6416 instruction set entirely, it is crucial to add the new specific instructions on the CPU core test. These modifications are described in the third chapter.

The chip support library (CSL) is used when testing the enhanced DMA (EDMA), Viterbi decoder coprocessor (VCP), turbo decoder coprocessor (TCP), multichannel buffered serial port (McBSP) and Timers. The CSL provides a C-language interface for configuring and controlling on-chip peripherals.

It is important to note that this program only provides a confidence check. It is not as comprehensive as the tests done at production, which thoroughly check the device’s logic, performance, and electrical parameters. This program is not capable of detecting all potential faults.

1.1 System Requirements

The following is required to use the TMS320C6416 POST:

- A suitable host PC to support TI tools
- Code Composer Studio 2.10 or later revisions
- Emulator tools (XDS510, etc.)
- A board with the TMS320C6416 processor to run the POST test

1.2 Test Structure

The test contains six modules:

- Module 1. `Chk6xTest()`

  Verifies the CPU instruction set. This module has eight assembly files to perform the CPU test:

1.	alu_64x.asm	Arithmetic operations
2.	alu40.asm	40-bit Arithmetic
3.	basic_64x.asm:	Basic operations
4.	bit.asm:	Bit management
5.	circular_64x.asm:	Circular addressing instructions
6.	cond.asm:	Branch and conditional instructions
7.	mult_64x:	Multiplier operations
8.	sat_64x:	Saturation instructions

- Module 2. `MemoryEdmaTest()`

  Verifies the enhanced DMA (EDMA) and on-chip memory by writing, moving, comparing and restoring internal data memory through the EDMA.

- Module 3. `VcpTest()`

  Checks the VCP on a 3GPP AMR type of frame

- Module 4. `TcpTest()`

  Checks the TCP on a 3GPP 128kbps type of frame

- Module 5. `McBspTest()`

  Verifies the McBSP operation by switching on the digital loopback bit in the serial port control register (SPCR)

- Module 6. `TimerTest()`

  Verifies the timers by starting a counter in a closed loop and by waiting until a timer interruption is generated. The value of the counter variable should be similar to the number of cycles executed in the test.

  As soon as one module fails, the test will skip the rest of the modules and will then enter an infinite loop, keeping the error value of the first failed module in a C variable called `Error` (see Figure 1).

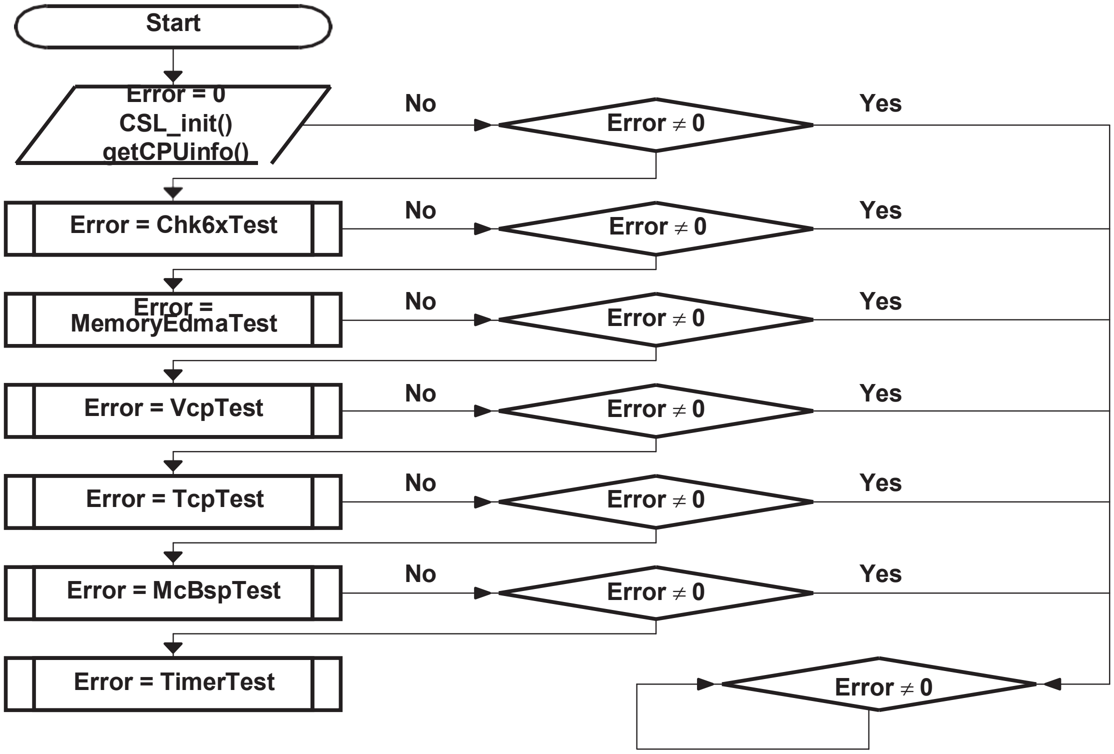


Figure 1. Main() Routine Flowchart

2 Modules Description

2.1 TMS320C64x Core Module

This module is based on the *TMS320C62x Self-Check Program* (SPRA635), which verifies the proper operation of the TMS320C62x instructions. As the TMS320C64x provides a superset of the TMS320C62x architecture, all the TMS320C62x instructions are still tested. Nevertheless, extensions are required to include the additional instructions of the TMS320C64x. These additional instructions are tested in alu_64x.asm, basic_64x.asm, mult_64x and sat_64x assembly files.

Table 1 shows the TMS320C64x specific instructions. For more details, see *TMS320C6000 CPU and Instruction Set* (SPRU189F).

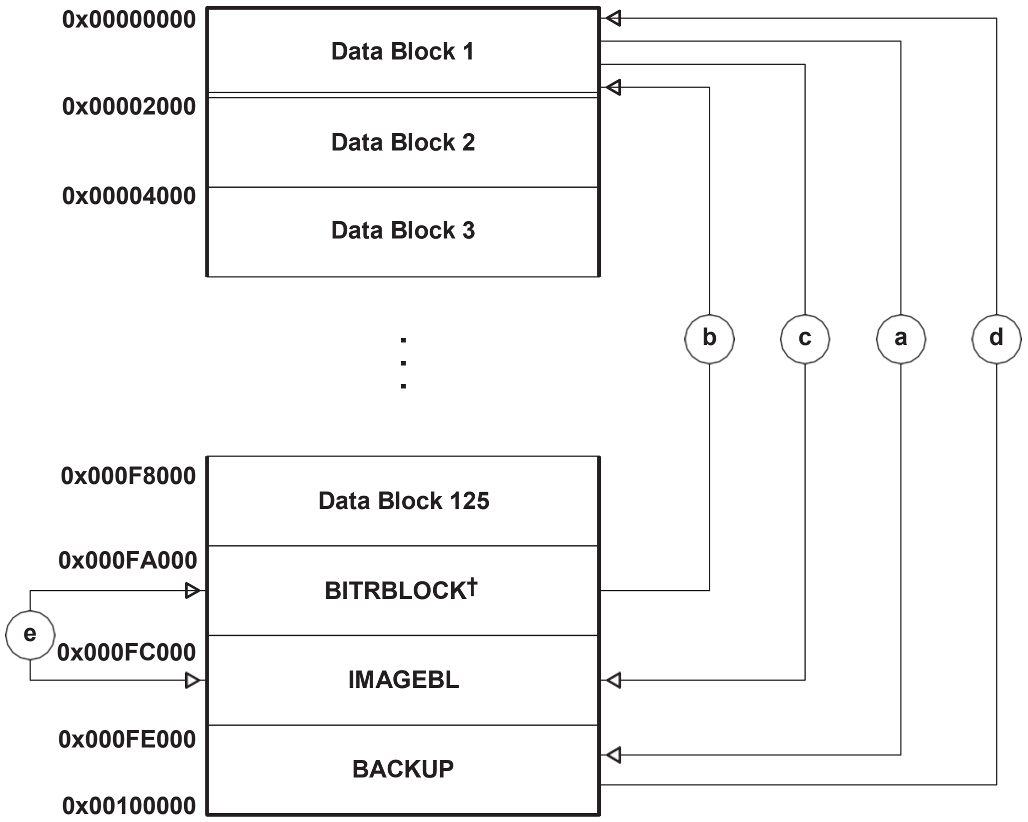
Table 1. C64x Specific Instructions by Functional Unit

.L Unit	.M Unit	.S Unit	.D Unit
ABS2	AVG2	ADD2	ADD2
ADD2	AVGU4	ADDKPC	ADDAD
ADD4	BITC4	AND	AND
AND	BITR	ANDN	ANDN
ANDN	DEAL	BDEC	LDDW
MAX2	DOTP2	BNOP	LDNDW
MAXU4	DOTPN2	BPOS	LDNW
MIN2	DOTPNRSU2	CMPEQ2	MVK
MINU4	DOTPNRUS2	CMPEQ4	OR
MVK	DOTPRSU2	CMPGT2	STDW
OR	DOTPRUS2	CMPGTU4	STDNW
PACK2	DOTPSU4	CMPL2	STNW
PACKH2	DOTPUS4	CMPLTU4	SUB2
PACKH4	DOTPU4	MVK	XOR
PACKHL2	GMPY4	OR	
PACKL4	MPY2	PACK2	
PACKLH2	MPYHI	PACKH2	
SHLMB	MPYIH	PACKHL2	
SHRMB	MPYHIR	PACKLH2	
SUB2	MPYIHR	SADD2	
SUB4	MPYLI	SADDU4	
SUBABS4	MPYIL	SADDSU2	
SWAP2	MPYLIR	SADDUS2	
SWAP4	MPYILR	SHLMB	
UNPKHU4	MPYSU4	SHR2	
UNPKLU4	MPYUS4	SHRMB	
XOR	MPYU4	SHRU2	

.L Unit	.M Unit	.S Unit	.D Unit
	MVD	SPACK2	
	ROTL	SPACKU4	
	SHFL	SUB2	
	SMPY2	SWAP2	
	SSHVL	UNPKHU4	
	SSHVR	UNPKLU4	
	XPND2	XOR	
	XPND4		

2.2 Internal Memory and EDMA Module

In addition to a memory buffer (0x000FA000-0x00100000), the internal memory is split into 125 blocks. The buffer is used to back up, write, read, and compare each block. Figure 2 describes the internal memory test process for data block 1. This process is repeated until all the125 blocks have been tested.



a Save Data Block 1 in the BACKUP space

b Write the data from BITRBLOCK to Data Block 1

c Read and write the data from Data Block 1 to IMAGEBL

d Restore the Data Block 1 values from the BACKUP memory

e Compare IMAGEBL with BITRBLOCK

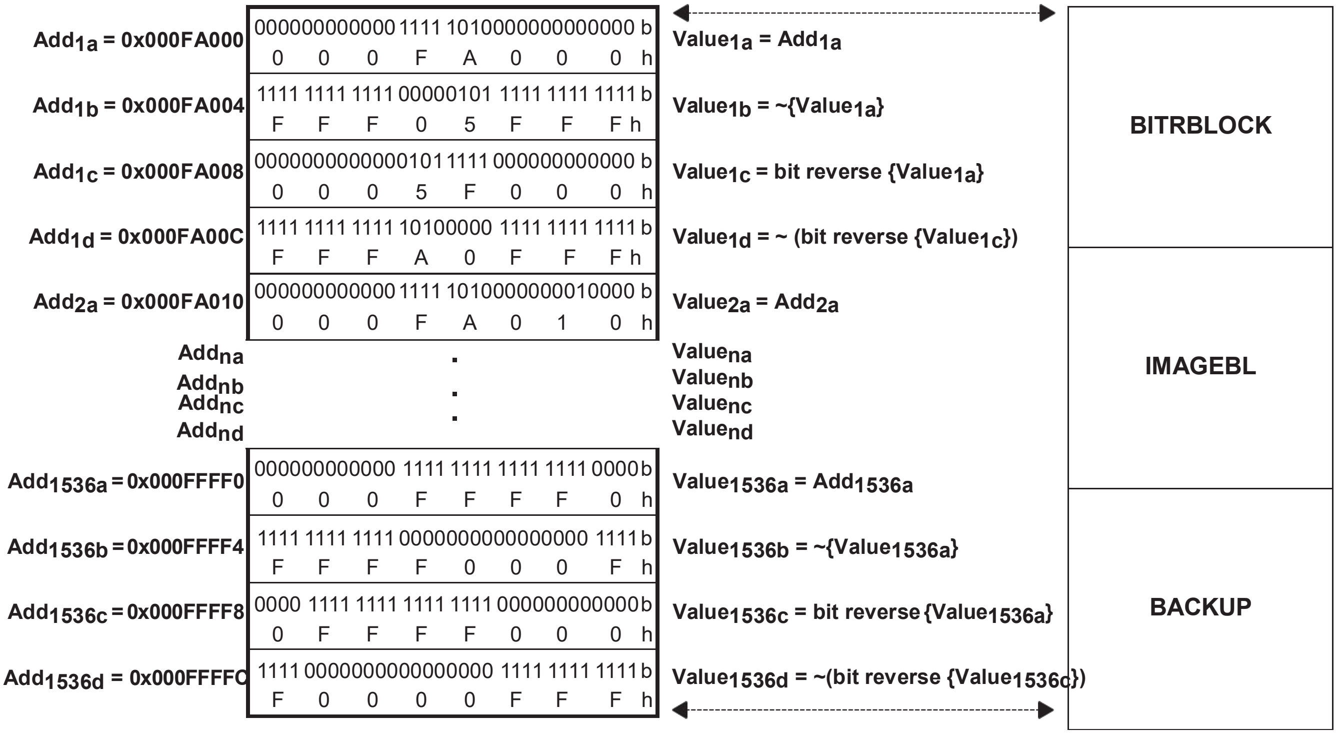
† BITRBLOCK is filled with generated data described later in this chapter (steps 1-5).

Figure 2. MemoryEdmaTest() Module Flowchart

The buffer plays a key role in the memory test, so it is important to first check the proper operation of the memory where the buffer resides. The procedure described below tests the reserved memory used as a buffer.

1. Save the 32 bits address number Add$_{na}$ (see Figure 3) at the address Add$_{na}$. When n equals 1, Add$_{1a}$ = 0x000FA000.
2. Invert the address number Add$_{na}$ and save it back to the address Add$_{nb}$ (Add$_{nb}$ = Add$_{na}$ +0x4). When n=1, Add$_{1b}$ = 0x000FA004.
3. Bit reverse the address number Add$_{na}$ and save it back to the address Add$_{nc}$ (Add$_{nc}$ = Add$_{na}$ +0x8). When n=1, Add$_{1c}$ = 0x000FA008.

4. Invert the previous bit-reversed word (specified in step 3) and save it back to the address Add$_{nd}$ (Add$_{nd}$ = Add$_{na}$ + 0xC). When n=1, Add$_{1d}$ = 0x000FA00C.
5. Repeat this process until the end of the buffer is reached (Add$_{1536}$).



Add$_{1a}$ = 0x000FA000
000000000000 1111 1010000000000000 b
0 0 0 F A 0 0 0 h
Value$_{1a}$ = Add$_{1a}$
Add$_{1b}$ = 0x000FA004
1111 1111 1111 00000101 1111 1111 1111 b
F F F 0 5 F F F h
Value$_{1b}$ = ~{Value$_{1a}$}
Add$_{1c}$ = 0x000FA008
0000000000000101 1111 000000000000 b
0 0 0 5 F 0 0 0 h
Value$_{1c}$ = bit reverse {Value$_{1a}$}
Add$_{1d}$ = 0x000FA00C
1111 1111 1111 10100000 1111 1111 1111 b
F F F A 0 F F F h
Value$_{1d}$ = ~ (bit reverse {Value$_{1c}$})
Add$_{2a}$ = 0x000FA010
000000000000 1111 1010000000010000 b
0 0 0 F A 0 1 0 h
Value$_{2a}$ = Add$_{2a}$
Add$_{na}$
Add$_{nb}$
Add$_{nc}$
Add$_{nd}$
Value$_{na}$
Value$_{nb}$
Value$_{nc}$
Value$_{nd}$
Add$_{1536a}$ = 0x000FFFF0
000000000000 1111 1111 1111 1111 0000b
0 0 0 F F F F 0 h
Value$_{1536a}$ = Add$_{1536a}$
Add$_{1536b}$ = 0x000FFFF4
1111 1111 1111 0000000000000000 1111b
F F F F 0 0 0 F h
Value$_{1536b}$ = ~{Value$_{1536a}$}
Add$_{1536c}$ = 0x000FFFF8
0000 1111 1111 1111 1111 000000000000b
0 F F F F 0 0 0 h
Value$_{1536c}$ = bit reverse {Value$_{1536a}$}
Add$_{1536d}$ = 0x000FFFFC
1111 0000000000000000 1111 1111 1111b
F 0 0 0 0 F F F h
Value$_{1536d}$ = ~(bit reverse {Value$_{1536c}$})
BITRBLOCK
IMAGEBL
BACKUP

Figure 3. Buffer Test

Once the buffer is filled, the next step is to recalculate each word and compare it to the values contained in the memory buffer. If the buffer values don't match the expected number, the test will output an error.

6. Compare the address number Add$_{na}$ (see Figure 3) with the value contained at the address Add$_{na}$ (Value$_{na}$). When n=1, Value$_{1a}$ = 0x000FA000.
7. Invert the address number Add$_{na}$ and compare it to the value contained at the address Add$_{nb}$ (Value$_{nb}$). When n=1, Value$_{1b}$ = 0xFFF05FFF.
8. Bit-reverse the address number Add$_{na}$ and compare it to the value contained at the address Add$_{nc}$ (Value$_{nc}$). When n=1 ⇒ Value$_{1c}$ = 0x0005F000.
9. Invert the previous bit-reversed word (specified in step 8) and compare it to the value contained at the address Add$_{nd}$ (Value$_{nd}$). When n=1 ⇒ Value$_{1d}$ = 0xFFFA0FFF.
10. Repeat this process until the end of the buffer is reached (Add$_{1536}$).

Note that there is an automatic process that maintains coherency between L1D cache and L2 SRAM when using the EDMA. For more details regarding this topic, please review *TMS320C6000 Peripherals* (SPRU190D) section 3.7.10, *EDMA Coherency*.

2.3 VCP Module

The VCP is tested by comparing known values with the result generated by the Viterbi coprocessor. The VCP outputs and inputs are sent and received through the EDMA.

Explanation of the Viterbi algorithm is beyond the scope of this document, but details can be found in the *Viterbi Decoder Coprocessor User's Guide* (SPRU533) and *Using TMS320C6416 Coprocessors: Viterbi Coprocessor (VCP)* (SPRA750).

The VCP base parameters structure used to set up the VCP programmable parameters is listed in Table 2.

Table 2. The VCP_BaseParams†

Code Rate	3
Constraint Length	9
Frame Length	81
Yamamoto Threshold	0
State Index	0
Hard/Soft Decision	0
Output Parameter Read Flag	1

† Variable located in vcp_parameters.h file.

The following steps roughly describe the VCP test (vcpTcp.c file).

1 Generate the VCP input configuration register. `VCPIC0-VCPIC5` registers are generated using the `VCP_BaseParams` variable (Table 2) in conjunction with `VCP_genParams` and `VCP_genIc` Application Programming Interfaces (APIs) of CSL.

2 Run the `VcpSubmitEdma` function. This will configure the EDMA to transmit[1] and receive[2] all the information required by the VCP. More information on how to program the VCP through the EDMA can be found in the *VCP User's Guide* (SPRU533), chapter 10.

NOTE 1: Input Configuration Registers and Branch Metrics.

NOTE 2: Decisions (the EDMA has programmed 4 different possibilities to receive data depending on whether it is a Hard or Soft Decision and whether the traceback mode is tailed or not) and output parameters data.

3 Execute the `VCP_Start` function included in the *Chip Support Library* (SPRU401D). This API will generate a VCP Transmit event[3] (VCPXEVT) that will trigger the EDMA and execute step 2.

NOTE 3: VCPREVT (EDMA channel 28) and VCPXEVT (EDMA channel 29) are used as synchronization event for EDMA transfer.

4 Check the decisions of the VCP. If the known values do not match the VCP output decisions, an error value is returned.

2.4 TCP Module

The TCP test works in a similar way as the VCP. The TCP is tested by comparing known values with the result generated by the Turbo coprocessor.

Explanation of the Turbo decoder process is beyond the scope of this document, but details can be found in the *Turbo Decoder Coprocessor User's Guide* (SPRU534) and *Using TMS320C6416 Coprocessors: Turbo Coprocessor (TCP)* (SPRA749).

The TCP base parameters structure used to set up the TCP programmable parameters is listed in Table 3.

Table 3. The TCP_BaseParams†

TCP Decoder Standard3GPP/IS2000	0
Code Rate	3
Frame Length	2576
Prolog Size	32
Maximum Iteration	8
SNR Threshold	0
Interleaver Flag	1
Output Parameter Read Flag	1

† Variable located in `tcp_parameters.h` file.

The following steps roughly describe the TCP test (vcpTcp.c file).

1 Generate the TCP input configuration register. `TCPIC0-TCPIC11` registers are generated using the `TCP_BaseParams` and `hXabData` variables in conjunction with `TCP_genParams` and `TCP_genIc` APIs of CSL.

2 Run the `TcpSubmitEdma` function. This will configure the EDMA to transmit[4] and receive[5] all the information required by the TCP. More information on how to program the TCP through the EDMA can be found in the *TCP User's Guide* (SPRU534), chapter 10.

NOTE 4: Input configuration parameters, systematic and parities data, and interleaver indexes data.

NOTE 5: Hard decisions and output parameters data.

3 Execute the `TCP_Start` function included in the *Chip Support Library* (SPRU401D). This API will generate a TCP Transmit event[6] (TCPXEVT) that will trigger the EDMA and execute step 2.

NOTE 6: TCPREVT (EDMA channel 30) and TCPXEVT (EDMA channel 31) are used as synchronization event for EDMA transfer.

4 Check the decisions of the TCP. If the known values don't match the TCP output decisions, an error value is returned.

2.5 McBSP Module

The McBSP module enables the Digital Loop Back bit in the SPCR register to simplify the test. Using the McBSP with the loopback enabled will not require an external input signal. This module tests the three McBSPs.

Below is a description of the current module:

1. The data 0xB16E5FFC is written into McBSP0.
2. The McBSP0 reads the 8 LSB of the previous data and stores it in a variable called `mcbspdata`.
3. The variable `mcbspdata` is written in McBSP1 and then it is read back through the same serial port. The read value is stored again in the same variable `mcbspdata`.
4. Step 3 is repeated when testing McBSP2.
5. The C variable `mcbspdata` must be compared with 0x000000FC.

2.6 Timer Module

This module tests the three timers.

The assembler routine `start_timer`, starts running a timer and a counter. The timer is configured to interrupt the routine every 100 times. As the TMS320C6416 internal timer input clock runs at CPU rate/8 (check *TMS320C6000 Peripherals Reference Guide*, Table 13-1) two is added to the counter variable `count` every 16 instructions.

When the timer counter register (CNT) register reaches 100, the interrupt is triggered and the program counter (PC) jumps to the vector table and then to the interrupt service routine (ISR) that closes the timer and sets a flag (`timer_done` =1) to exit the `start_timer` routine.

If the counter is between 95 and 104 it is assumed that the timer is working correctly.

This process is repeated for Timer0, Timer1 and Timer2.

3 Changing the TMS320C6416 POST Endianess

The steps below are needed to change the POST endianess.

1. Change the compiler option to the correct endianess. Project →Build Options…→Compiler tab→Advanced category→Endianess→`Little Endian` or `Big Endian` (-me)
2. Change the included libraries in the linker option. Project →Build Options…→Linker tab→Basic category→Include Libraries (-l) → “`csl6416.lib;rts6400.lib`” for Little Endian or “`csl6416e.lib;rts6400e.lib`” for Big Endian.
3. Rebuild the POST.

Note that the test relies on the predefined symbolic constants: LITTLE_ENDIAN and BIG_ENDIAN.

4 Error Codes

Table 4 lists all possible error codes that the POST can return. It also gives the name of the module that generates the error code. It is important to note that the code descriptions identify only potential causes of the error. They should not be taken as absolute. Any number of actual malfunctions could generate a particular error code. For example, a bad memory location would cause every test using it to fail.

Table 4. Error Codes

Code	Description	Source File
11h	LOAD instruction error [LD(B/BU)(H/HU)(W)]	basic_64x
12h	STORE instruction error [ST(B/BU)(H/HU)(W)]	basic_64x
13h	MV and MVC instructions error	basic_64x
14h	ZERO instruction error	basic_64x
15h	MVK, MVKH, MVKLH instructions error	basic_64x
21h	Shift and CMP instructions error	alu_64x

Table 4. Error Codes (Continued)

Code	Description	Source File
22h	Logical instructions error (OR, XOR, AND etc.)	alu_64x
23h	Addition instructions error (ADDU, ADDK, ADD2 etc.)	alu_64x
24h	Subtraction instructions error (SUB, SUBA etc.)	alu_64x
25h	SUBC instruction error	alu_64x
26h	SUB2 instruction error	alu_64x
9F001h	ABS2 FAIL CODE	alu_64x
9F002h	ADD4 FAIL CODE	alu_64x
9F003h	ANDN FAIL CODE	alu_64x
9F004h	MAX2 FAIL CODE	alu_64x
9F005h	MAX4 FAIL CODE	alu_64x
9F006h	MIN2 FAIL CODE	alu_64x
9F007h	MINU4 FAIL CODE	alu_64x
9F008h	PACKH4 FAIL CODE	alu_64x
9F009h	PACKL4 FAIL CODE	alu_64x
9F00Ah	PACKL4 FAIL CODE	alu_64x
9F00Bh	PACKH2 FAIL CODE	alu_64x
9F00Ch	PACK2 FAIL CODE	alu_64x
9F00Dh	PACKL4 FAIL CODE	alu_64x
9F00Eh	PACKHL2 FAIL CODE	alu_64x
9F00Fh	PACKLH2 FAIL CODE	alu_64x
9F010h	SHLMB FAIL CODE	alu_64x
9F011h	SHRMB FAIL CODE	alu_64x
9F012h	SUB2 FAIL CODE	alu_64x
9F013h	SUB4 FAIL CODE	alu_64x
9F014h	SUBABS4 FAIL CODE	alu_64x
9F015h	SWAP2 FAIL CODE	alu_64x
9F016h	SWAP4 FAIL CODE	alu_64x
9F017h	UNPKHU4 FAIL CODE	alu_64x
9F018h	UNPKLU4 FAIL CODE	alu_64x
9F047h	ADD2 .S2 FAIL CODE	alu_64x

Table 4. Error Codes (Continued)

Code	Description	Source File
9F048h	ADD2 .D2 FAIL CODE	alu_64x
9F049h	SUB2 .S2 FAIL CODE	alu_64x
9F04Ah	SUB2 .D2 FAIL CODE	alu_64x
9F04Bh	PACK2 .S2 FAIL CODE	alu_64x
9F04Ch	PACKH2 .S2 FAIL CODE	alu_64x
9F04Dh	PACKHL2 .S2 FAIL CODE	alu_64x
9F04Eh	PACKLH2 .S2 FAIL CODE	alu_64x
9F053h	SHLMB .S2 FAIL CODE	alu_64x
9F054h	SHRMB .S2 FAIL CODE	alu_64x
9F055h	SHR2 .S2 FAIL CODE	alu_64x
9F056h	SHRU2 .S2 FAIL CODE	alu_64x
9F057h	SWAP2 .S2 FAIL CODE	alu_64x
9F058h	UNPKHU4 .S2 FAIL CODE	alu_64x
9F059h	UNPKLU4 .S2 FAIL CODE	alu_64x
9F05Ah	SPACK2 .S2 FAIL CODE	alu_64x
9F05Bh	SPACKU4 .S2 FAIL CODE	alu_64x
9F05Ch	OR .L2 FAIL CODE	alu_64x
9F05Dh	OR .S2 FAIL CODE	alu_64x
9F05Eh	OR .D2 FAIL CODE	alu_64x
9F05Fh	XOR .S2 FAIL CODE	alu_64x
9F060h	XOR .D2 FAIL CODE	alu_64x
9F061h	MVK .L2 FAIL CODE	alu_64x
9F062h	MVK .S2 FAIL CODE	alu_64x
9F063h	MVK .D2 FAIL CODE	alu_64x
9F064h	AND .L2 FAIL CODE	alu_64x
9F065h	AND .S2 FAIL CODE	alu_64x
9F066h	AND .D2 FAIL CODE	alu_64x
9F067h	ANDN .S2 FAIL CODE	alu_64x
9F068h	ANDN .D2 FAIL CODE	alu_64x
9F069h	ADDAD .D2 FAIL CODE	alu_64x

Table 4. Error Codes (Continued)

Code	Description	Source File
9F06Fh	LDDW & STDW .D2 FAIL CODE	alu_64x
9F070h	LDDW & STDW .D2 FAIL CODE	alu_64x
9F071h	LDNDW & STNDW .D2 FAIL CODE	alu_64x
9F072h	LDNDW & STNDW .D2 FAIL CODE	alu_64x
9F073h	LDNW & STNW .D2 FAIL CODE	alu_64x
9F074h	CMPEQ2 .S2 FAIL CODE	alu_64x
9F075h	CMPGT2 & STNW .D2 FAIL CODE	alu_64x
9F076h	CMPGTU4 .S2 FAIL CODE	alu_64x
9F077h	ADKPC .S2 FAIL CODE	alu_64x
9F078h	BDEC .S2 FAIL CODE	alu_64x
9F079h	BPOS if branch not taken to Test4 .S2 FAIL CODE	alu_64x
9F07Ah	BPOS if branch taken to Test4 .S2 FAIL CODE	alu_64x
31h	MPY instruction error	mult_64x
32h	MPYH, MPYHUS instructions error	mult_64x
33h	MPYHU and MPYHSU instructions error	mult_64x
34h	MPYHL instruction error	mult_64x
35h	MPYLH instruction error	mult_64x
9F019h	AVG2 Fail code	mult_64x
9F01Ah	AVGU4 Fail code	mult_64x
9F01Bh	BITC4 Fail code	mult_64x
9F01Ch	BITR Fail code	mult_64x
9F01Dh	DEAL Fail code	mult_64x
9F01Eh	DOTP2 Fail code	mult_64x
9F01Fh	DOTPN2 Fail code	mult_64x
9F020h	DOTPNRSU2 Fail code	mult_64x
9F021h	DOTPNRUS2 Fail code	mult_64x
9F022h	DOTPRUS2 Fail code	mult_64x
9F023h	DOTPRUS2 Fail code	mult_64x
9F024h	DOTPSU4 Fail code	mult_64x
9F025h	DOTPUS4 Fail code	mult_64x

Table 4. Error Codes (Continued)

Code	Description	Source File
9F026h	DOTPU4 Fail code	mult_64x
9F027h	GMPY4 Fail code	mult_64x
9F028h	GMPY4 Fail code	mult_64x
9F029h	MPY2 Fail code	mult_64x
9F02Ah	MPY2 Fail code	mult_64x
9F02Bh	MPYHI Fail code	mult_64x
9F02Ch	MPYHI Fail code	mult_64x
9F02Dh	MPYIH Fail code	mult_64x
9F02Eh	MPYIH Fail code	mult_64x
9F02Fh	MPYHIR Fail code	mult_64x
9F030h	MPYIHR Fail code	mult_64x
9F031h	MPYLI Fail code	mult_64x
9F032h	MPYLI Fail code	mult_64x
9F033h	MPYIH Fail code	mult_64x
9F034h	MPYIH Fail code	mult_64x
9F035h	MPYLIR Fail code	mult_64x
9F036h	MPYILR Fail code	mult_64x
9F037h	MPYSU4 Fail code	mult_64x
9F038h	MPYSU4 Fail code	mult_64x
9F039h	MPYUS4 Fail code	mult_64x
9F03Ah	MPYUS4 Fail code	mult_64x
9F03Bh	MPYU4 Fail code	mult_64x
9F03Ch	MPYU4 Fail code	mult_64x
9F03Dh	MVD Fail code	mult_64x
9F03Eh	ROTL Fail code	mult_64x
9F03Fh	SHFL Fail code	mult_64x
9F040h	SMPY2 Fail code	mult_64x
9F041h	SMPY2 Fail code	mult_64x
9F042h	SSHVL Fail code	mult_64x
9F043h	SSHVR Fail code	mult_64x

Table 4. Error Codes (Continued)

Code	Description	Source File
9F044h	XPND2 Fail code	mult_64x
9F045h	XPND4 Fail code	mult_64x
41h	CLR instruction error BIT	bit
42h	EXT instruction error [EXT(U)]	bit
43h	LMBD instruction error	bit
44h	NORM instruction error	bit
45h	SET instruction error	bit
51h	SSHL instruction error	sat_64x
52h	SADD instruction error	sat_64x
53h	SAT instruction error	sat_64x
54h	SSUB instruction error	sat_64x
55h	SMPY(L)(H) instructions error	sat_64x
56h	SMPYHL instruction error	sat_64x
9F04Fh	SADD2 error	sat_64x
9F050h	SADDU4 error	sat_64x
9F051h	SADDUS2 error	sat_64x
9F052h	SADDSU2 error	sat_64x
61h	B instruction [Conditional/Unconditional] error	cond
62h	Conditional Instructions error	cond
71h	ADDAB instruction error	circular_64x
72h	ADDAH instruction error	circular_64x
73h	ADDAW instruction error	circular_64x
74h	CIRCULAR BUFFER AUTHENTICITY error (LDW)	circular_64x
81h	ADDU instruction (for 40 bit) error	alu40
82h	CMPEQ instruction (for 40 bit) error	alu40
83h	SUBU instruction (for 40 bit) error	alu40
01h	Memory Test failed	memory_edma
02h	Edma Channel Allocation failed	memory_edma
04h	VCP Edma Channel Allocation failed	vcpTcp
05h	TCP Edma Channel Allocation failed	vcpTcp

Table 4. Error Codes (Continued)

Code	Description	Source File
08h	VCP decoding failed	vcpTcp
09h	TCP decoding failed	vcpTcp
10h	McBsp operation failed	mcbsptest
20h	Timer0 operation failed	timer
40h	Timer1 operation failed	timer
80h	Timer2 operation failed	timer

5 References

1. *TMS320C62x Self-Check Program Application Brief* (Literature number SPRA635)
2. *TMS320C6000 CPU and Instruction Set (Rev. F)* (Literature number SPRU189F)
3. *TMS320C6000 Peripherals Reference Guide (Rev. D)* (Literature number SPRU190D)
4. *TMS320C6000 Chip Support Library API Reference Guide (Rev. D)* (Literature number SPRU401D)
5. *Viterbi Decoder Coprocessor User’s Guide* (Literature number SPRU533)
6. *Turbo Decoder Coprocessor User’s Guide* (Literature number SPRU534)
7. *Using TMS320C6416 Coprocessors: Viterbi Coprocessor (VCP)* (Literature number SPRA750)
8. *Using TMS320C6416 Coprocessors: Turbo Coprocessor (TCP)* (Literature number SPRA749)

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265