

# TMS320C28x INSTRUCTION SET SIMULATOR TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

- Included in Code Composer Studio™ Integrated Development Environment (IDE) for TMS320C2000™
- TMS320C28x™ CPU Full Instruction Set Architecture Execution
  - Peripheral Interrupt Expansion (PIE)
  - VBUS
  - Timer
  - Flash Memory
- Configurable Memory Simulation
- Accurate Cycle Simulation
  - On-Chip Memory Blocks
  - External Memory Blocks
- Port Connect
  - Supports External Data Simulation
- Pin Connect
  - Supports External Event Simulation
- Supports Pipeline Display

## Description

The TMS320C28x Instruction Set Simulator, available within the Code Composer Studio™ for TMS320C2000™ Integrated Development Environment (IDE), simulates the instruction set of the C28x™ core. Table 1 lists the simulator cores and peripherals supported, with the corresponding configuration to be selected under the Import Configuration menu of Code Composer Studio Setup.

**Table 1. Processors Supported by the C28x Simulator**

PROCESSOR	CODE COMPOSER STUDIO IDE IMPORT CONFIGURATION
TMS320F28x (CPU only)	F28xx Cycle Accurate Simulator
TMS320F2810 (CPU + VBUS, PIE, Timer, Flash)	F2810 Device Simulator
TMS320F2812 (CPU + VBUS, PIE, Timer, Flash)	F2812 Device Simulator
TMS320F2810 and TMS320F2812 Tutorial	F28xx Simulator Tutorial on input/output ports, Timer and VBUS

Code Composer Studio, TMS320C2000, TMS320C28x, and C28x are trademarks of Texas Instruments.  
All trademarks are the property of their respective owners.



Copyright © 2003, Texas Instruments Incorporated

# TMS320C28x INSTRUCTION SET SIMULATOR TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

---

## Supported Hardware Resources

### CPU

The C28x simulator simulates the full C28x instruction set architecture (except emulation instructions).

### Memory

The simulator provides configurable memory simulation. By default, the simulator does not provide any memory mapping specific to the device or processor. All the memory blocks in Program, Data, and I/O space can be simulated by adding memory blocks using the simulator configuration file. The memory blocks can also be added using the "Memory Map Add" feature in GEL for debugger visibility.

The simulator provides cycle accurate simulation for both core and memory blocks. For example, for on-chip memories, the simulator takes care of the number of cycles required to access memory depending on wait states specified by the simulator configuration file. For more details, refer to the section on Configuring the Simulator, starting on page 7.

### Peripherals

The F2810 and F2812 simulators model the following peripherals:

- PIE - Peripheral Interrupt Expansion Block
- Flash
- Timer
- VBUS

The PIE, Timer and Flash memory are available as add-on modules, while VBUS is part of the C28x simulator core. In order to simulate a peripheral, it must be included in the simulator configuration file. For more details, refer to the section on Peripherals, beginning on page 13.

### Block Protection

This feature adds write followed by read pipeline protection for a block of memory. Any read to the memory bank will stall until the write is completed. For more details on Block Protection, see the *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430). The user can select an area in the memory to pipeline protect using PROTSTART, PROTRANGE and ECNF memory mapped registers.

**Table 2. Block Protection on C28x**

NAME	ADDRESS	SIZE	DESCRIPTION
ECNF	0x880	16	Setting the seventh (7th) bit to one (1) enables block protection
PROTSTART	0x884	16	Start address
PROTRANGE	0x885	16	Range in units of 64 words: 0000 means 64 words 0001 means 128 words, etc.

## Supported Simulation Features

### External Event Simulation

The simulators simulate the hardware inside a particular DSP device, whereas in real hardware DSP interacts with many other external entities. The interactions between the simulator and these external entities fall into the following two categories:

- **Control Signals** - These signals trigger activities to the simulator (e.g. interrupts, serial port clocks, serial port synchronization events, etc.)
- **Data Values** - These are part of an interaction between the simulator and an external entity (e.g. read and writes to peripheral registers as a part of I/O memory, serial port data, etc.)

For example, in an audio device the serial port of the DSP is connected to A/D and D/A converters or to a codec. The interaction between the DSP and the audio device happens through transfer of a synchronization signal to start a sample, as well as the sample data itself. Here the synchronization signal falls into the Control Signals category and the sample data falls in Data Values category.

The simulator provides two features - namely **Pin Connect** and **Port Connect** - for the simulation of these two types of interactions, respectively.

### Pin Connect

The Pin Connect tool allows the user to simulate events from the external world. Generally, control signals from external entities to the simulator are of most interest to the user. Pin Connect provides a generic way to simulate the control signals from the external entities to the simulator. In these cases, only the control signals and the time at which the signal must be triggered are important. The simulator provides the user with a list of pins corresponding to different control signals. The user must specify the clock values at which events are to be triggered on this pin using a special format in a file. The user must then connect this file to the pin using command window, GEL commands or Pin Connect plug-in. For more details, refer to the section titled Pin Connect File Format, beginning on page 10.

The C28x simulator provides the Pin Connect feature for all processor configurations. The list of pins supported is shown in Table 3.

**Table 3. Pins Supported by the Pin Connect Feature in the C28x Simulator**

INT1
INT2
INT3
INT4
INT5
INT6
INT7
INT8
INT9
INT10
INT11
INT12
INT13
INT14
DLOGINT
RTOSINT
NMI
EMUINT

# TMS320C28x INSTRUCTION SET SIMULATOR TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

## Port Connect

The Port Connect feature allows the user to simulate data transfer between the DSP and an external entity which is present in the real hardware. The transfer of data between the DSP and an external entity (which sits at some particular address in the memory space of the DSP) can happen in the following two ways:

- Data from an external entity to the DSP
- Data from the DSP to an external entity

To simulate data transfer from an external entity to the DSP, first, all data values which will be transferred from the external entity to the DSP are put into a file (the details of the file format are discussed later). Then an association is made in the simulator between this file and the address at which the external entity sits. This association is called read-mode port-connect. Whenever the simulator must read from the external entity through the address associated, it reads from the file one word at a time. To simulate the data transfer from the DSP to an external entity, a file is port-connected in write mode against the address of the external entity. Whenever the simulator has to write data to that address, it writes in the file one word at a time.

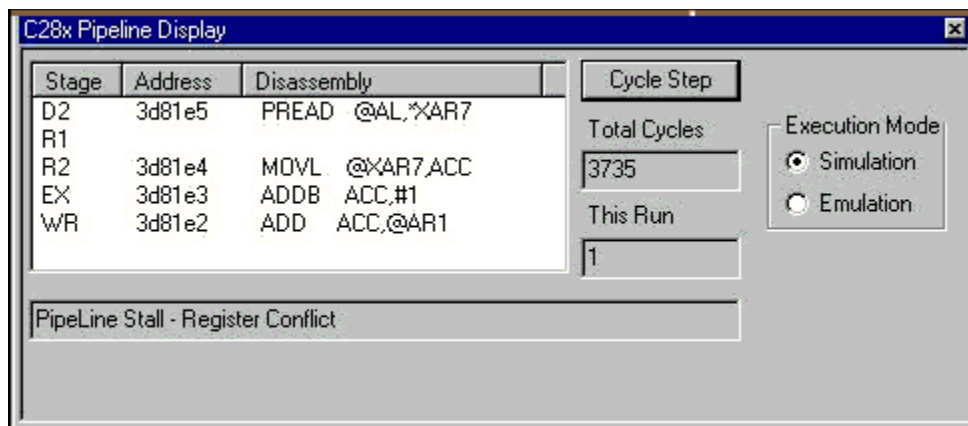
The C28x simulator provides the Port Connect feature for Program, Data, and I/O space for all processor configurations. The simulator has special instructions, such as IN, OUT, and UOUT for performing I/O reads and writes. Before using this feature, make sure that the address being connected to is already mapped in the simulator configuration file. This is important in case of address in I/O, as the simulator by default does not include those addresses in its memory map. Please note that in the case of serial ports, data can be transmitted by connecting some files at the memory mapped locations for the serial port transmit register in write mode. Similarly, data can be received by connecting some files at the memory mapped locations for serial port receive register in read mode.

## Pipeline Display

C28x is a pipeline-protected device. Although the hardware guarantees correct execution by introducing stall cycles, the result may be more execution cycles than necessary, making the program less efficient in terms of performance. Also, the maximum optimization level possible is currently limited by compiler capabilities. After the maximum optimization is done by the compiler DSP application, further optimization is challenging. Like the hardware, the C28x simulators introduce stall cycles to avoid data hazards. Since the simulator is a software representation of the device, it is possible to use its flexibility to develop more efficient code. The C28x Pipeline Display plug-in helps the C28x programmer to write more efficient DSP code by avoiding unnecessary stall cycles.

Figure 1 shows the five stages of the C28x pipeline display plug-in and indicates if the pipeline has been stalled due to a register or memory conflict.

**Figure 1. C28x Pipeline Display Plug-In Window**



## Simulation and Emulation Mode

### *Simulation Mode Overview*

Simulation mode is the default execution mode in the C28x simulator. In this mode, the instructions are executed in a pipelined fashion, as in the actual hardware. Stalls are introduced in the pipeline to avoid data hazards.

### *Emulation Mode Overview*

In emulation mode, the instructions are executed one at a time. An instruction is taken through all the phases of the pipeline before the next instruction is allowed to enter the pipeline. This mode makes debugging easier, since it eliminates all pipeline effects and helps the user to concentrate on the functionality of the instruction. In emulation mode, the pipeline plug-in displays only the one instruction that is about to be executed.

### *Switching Between Modes*

Simulation mode is the default mode. You can switch between modes using either of the following methods:

- TI Command Window plug-in - Command "EMU" will take you to emulation mode and "SIM" will take you to simulation mode.
- Pipeline Plug-in - Use the radio button "Execution Mode" to select the mode.


#### **Notes:**

- Cycle step is disabled under emulation mode.
- Profiling in emulation mode will give incorrect results.

# TMS320C28x INSTRUCTION SET SIMULATOR TECHNICAL OVERVIEW

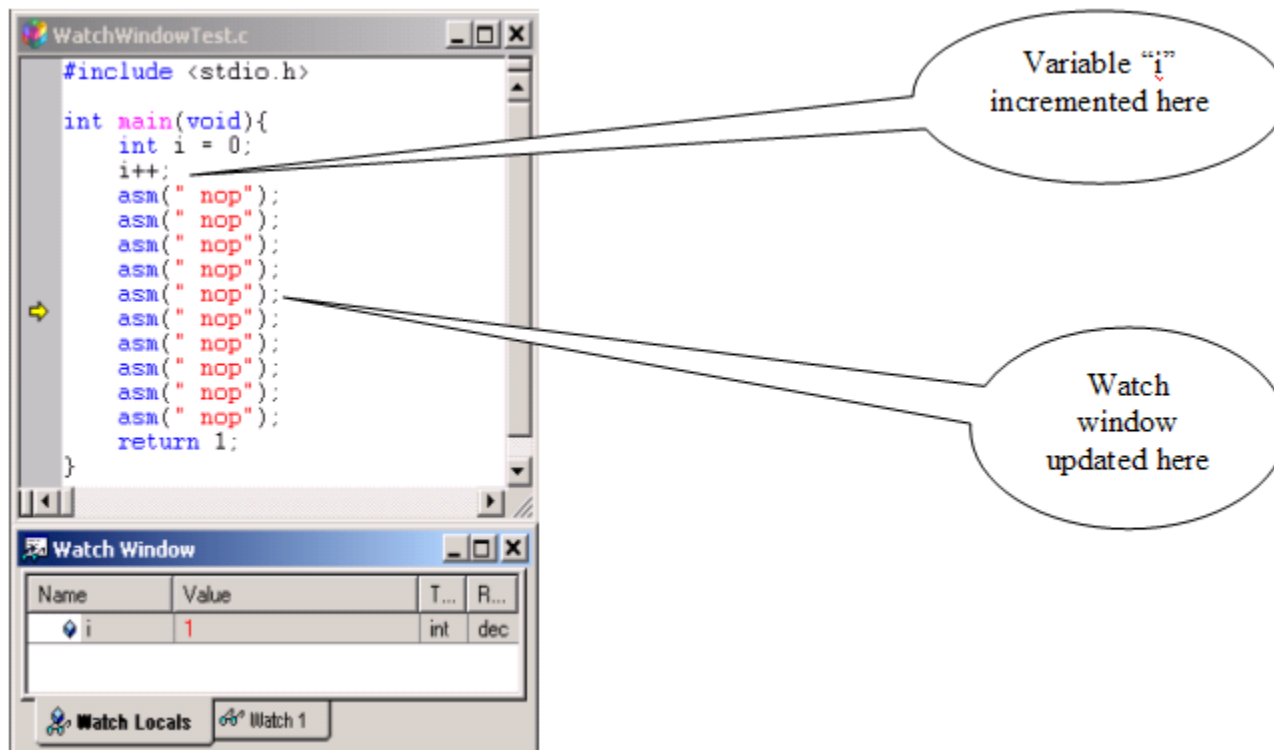
SPRU608B – JULY 2002 – REVISED OCTOBER 2003

## Pipeline Effects on Watch Window Display

In C28x simulators, the debug arrow (  ) points to the instruction which is in the D2 stage of the pipeline. A memory write due to this instruction will happen only when the instruction completes its WR phase. This will happen after a few instruction steps.

This pipeline effect causes problems in watch window updates. If the user sets up a variable in the watch window and steps over a code which updates that variable, the watch window may not show the new value immediately after the instruction step. Instead this will happen after a few instruction steps when the instruction actually completes its WR phase. This causes confusion in debugging. Consider the following example:

Figure 2. Watch Window Problem



In the figure above, notice that even though the variable "i" was incremented earlier, the update (indicated by red color) in the watch window happens later.

Such effects do not exist in Emulation mode, as an instruction step causes all the stages of the instruction to be executed before the next instruction enters the pipeline.

### General Tip:

- Use Simulation Mode for Application Code Profiling and Tuning.
- Use Emulation Mode for Application Debug.

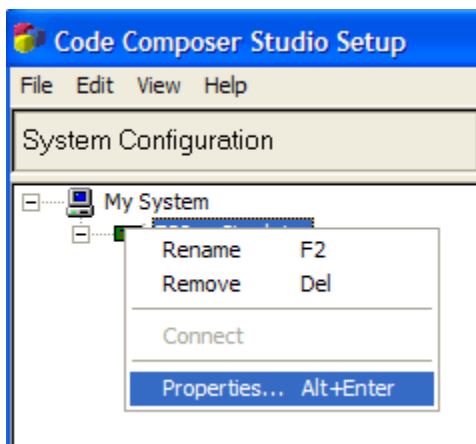
## Configuring the Simulator


### Simulator Configuration File

The simulator configuration file contains information about the memory map, peripherals and other configurable properties of the simulator. To view the simulator configuration file for a given simulator target, perform the following steps:

1. Invoke Code Composer Studio Setup.
2. In the "System Configuration" tab, right click on the simulator and select properties (see Figure 3).

**Figure 3. Code Composer Studio System Configuration**



3. In the pop-up window that appears, select the "Board Properties" tab.
4. Double-click the browse button (  ) at the end of the Value column to open the path to the simulator configuration file.

The simulator configuration files are located in <CCS\_INSTALLATION\_DIR>/drivers. Table 4 lists the names of simulator configuration files for the various C28x simulators.

**Table 4. Simulator Configuration Files for C28x Simulator Configurations**

CODE COMPOSER STUDIO IDE IMPORT CONFIGURATION	SIMULATOR CONFIGURATION FILE
F28xx Cycle Accurate Simulator	sim28xx.cfg
F2810 Device Simulator	sim2810.cfg
F2812 Device Simulator	sim2812.cfg
F28xx Simulator Tutorial	sim28xxtutorial.cfg

# TMS320C28x INSTRUCTION SET SIMULATOR

## TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

### Syntax of Simulator Configuration Files

#### Memory Maps

This section describes the syntax for specifying memory maps in the simulator configuration file. A description of keywords is found in Table 5.

```
module <module_name>;  
    // Comment  
    memory <memory_name>;  
        first      <address>;  
        last       <address>;  
        space      <space_type>;  
        waitstates <count>;  
        type       <memory_type>;  
        shared     <shared_memory_name>;  
    end <memory_name>;  
end <module_name>;
```

**Table 5. Keyword Description for Specifying Memory Maps in Simulator Configuration Files**

KEYWORD	ARGUMENT	DESCRIPTION
MODULE	<module_name>	Begin a module description,
MEMORY	<memory_name>	Begin the description of a block of memory,
END	<memory/module_name>	End the description of the named module or memory block,
FIRST	<address>	Specifies the starting address of the memory block,
LAST	<address>	Specifies the last address in the memory block. Memory block lengths must be even.
WAITSTATES	<count>	Specifies the number of wait states for a memory block. Defaults to 0.
SPACE	<space_type> Can be one of the following: <ul style="list-style-type: none"><li>● PROG</li><li>● DATA</li><li>● IOPORT</li><li>● OUTPORT</li></ul>	Specifies whether the memory block is a part of program or data space, or IO space.
TYPE	<memory_type> Can be one of the following: <ul style="list-style-type: none"><li>● SARAM</li><li>● ROM</li><li>● Flash</li></ul>	Indicates the nature of the memory. Flash memory in the simulator is treated like ROM. The Flash memory supported here is different from the Flash memory available as a peripheral module.
SHARED	<shared_memory_name>	Specifies that the block is shared with another named memory block. The shared memory blocks map to the same physical block of memory. A write to one will appear in the other. The SHARED keyword need not be specified in the other block.



## Memory Maps (continued)

For example:

```
module my_memory;
    // M0 SARAM, shared prog and data space
    memory M0_prog_mem;
        first      0x000000;
        last       0x0003FF;
        space      prog;
        waitstates 0;
        type       saram;
        shared     M0_data_mem;
    end M0_prog_mem;

    memory M0_data_mem;
        first      0x000400;
        last       0x0007FF;
        space      data;
        waitstates 0;
        type       saram;
    end M0_data_mem;

end my_memory;
```

In the above example, the program memory range from 0x0 to 0x3FF and the data memory range from 0x400 to 0x7FF map to the same block in the physical memory. Any write to this data memory will have an effect on the program memory, and vice-versa.

### Notes:

- Keywords are case insensitive.
- Each line must end with a ';' delimiter.
- Comments start with "///" and end in a new line.
- When specifying a memory block, you must specify the MEMORY <name>, the END <name>, FIRST and LAST addresses, the SPACE, and the TYPE of memory. All other fields are optional.
- <address> may be specified in hex or decimal.
- If the Flash memory type is specified, then it will be treated like ROM. The SPACE qualifier needs to be specified, as do the WAITSTATES. The default value of WAITSTATES is "0". The Flash behaves like a normal wait stated ROM without any additional functionality as described in the Flash hardware specification. To model this, the Flash peripheral module should be used.
- The Flash memory (OTP and Flash memory) can be specified either in internal memory or in external Flash peripheral module. It should not be specified in both.

# TMS320C28x INSTRUCTION SET SIMULATOR TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

## Peripheral Modules

This section describes the syntax for specifying peripheral modules in simulator configuration files. A description of keywords is found in Table 6.

```
Module main;
    cssi_modules <module_list>;
end main;

module <module_name>;
    cssi_library <relative_path_of_dll>;
    init_function <initialization_function>;
    <module details>
end <module_name>;
```

**Table 6. Keyword Description for Specifying Peripherals in Simulator Configuration Files**

KEYWORD	ARGUMENT	DESCRIPTION
MODULE	MAIN	Indicates that this module contains the names of peripherals to be hooked up.
CSSI_MODULES	<module_list>	Specifies a list of peripheral modules.
CSSI_LIBRARY	<relative_path_of_dll>	Specifies the name of the library to be opened by the simulator to execute to the peripheral. This keyword must be used inside a peripheral module description.
INIT_FUNCTION	<initialization_function>	Specifies the entry point to the library. This keyword must be used inside a peripheral module description.

For example:

```
Module main;
    cssi_modules Pie;
end main;

module Pie;
    cssi_library Pie.dll;
    init_function initPIE;
    INT3      ..\tutorial\sim28xx\Pie\int3.txt ;
    INT9      ..\tutorial\sim28xx\Pie\int9.txt ;
    INT27     ..\tutorial\sim28xx\Pie\int27.txt ;
    INT81     ..\tutorial\sim28xx\Pie\int81.txt ;
end Pie;
```

## File Format for Pin Connect

The simulator allows the user to simulate and monitor external interrupt signals. The Pin Connect tool enables the user to specify the interval at which selected external interrupts will occur.

To simulate external interrupts, follow the steps below:

1. Create a data file that specifies interrupt intervals.
2. Start the Pin Connect tool by choosing Pin Connect from the Code Composer Studio Tools menu.
3. Connect the data file to an external interrupt pin.
4. Load the program.
5. Run the program.



## Setting up the Input File

To simulate external interrupts, the user must first create a data file that specifies interrupt intervals. Interrupt intervals are expressed as a function of CPU clock cycles. Simulation begins at the first clock cycle. An interrupt will occur at each specified clock cycle.

The data file must contain a CPU clock cycle parameter in the following format:

```
clock-cycle [rpt {n|EOS}]
```

- **clock-cycle** - The CPU clock-cycle parameter specifies the intervals at which interrupts will occur. Clock cycles can be specified as absolute or relative.
- **rpt** - Repeat the same pattern a fixed number of times.
- **n** - A positive integer value specifying the number of times to repeat.
- **EOS** - Repeat the same pattern until the end of simulation.

## Absolute Clock Cycle

To use an absolute clock cycle, the cycle value must represent the actual CPU clock cycle where an interrupt should be simulated. For example,

```
12 34 56
```

Interrupts are simulated at the 12th, 34th, and 56th CPU clock cycles. No operation is performed on the clock cycle value; the interrupt occurs exactly as the clock cycle value is written.

## Relative Clock Cycle

The user can also select a clock cycle that is relative to the time at which the last event occurred. A plus sign (+) before a clock cycle adds that value to the total clock cycles proceeding it. For example,

```
12 +34 55
```

In this example, a total of three interrupts are simulated at the 12th, 46th (12 + 34), and 55th CPU clock cycles. The user can mix both relative and absolute values in the data file.

## Repetition of Patterns for a Specified Number of Times

The user can format the data file to repeat a particular pattern for a fixed number of times. For example,

```
5 (+10 +20) rpt 2
```

The values inside the parenthesis represent the portion that is repeated. Therefore, an interrupt is simulated at the fifth CPU cycle, then the 15th (5+10), 35th (15+20), 45th (35+10), 65th (45+20) CPU clock cycles.

## Repetition to the End of Simulation (EOS)

To repeat the same pattern throughout the simulation, add the string EOS to the line. For example,

```
10 (+5 +20) rpt EOS
```

Interrupts are generated at the tenth CPU cycle, the 15th (10+5), the 35th (15+20), the 40th (35+5), the 60th (40+20), and so on, continuing in that pattern until the end of simulation.

# TMS320C28x INSTRUCTION SET SIMULATOR

## TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

---

### File Format for Port Connect

The Port Connect file contains one or more lines. Each line contains less than 80 characters to represent one data value. The data is specified in hex format without any '0x' prefix or 'h' suffix. The following is a sample Port Connect file:

```
6666
9999
aaaa
cccc
7f7f
```

#### Notes:

- The first value is taken as  $(6666)_{16}$  not  $(6666)_{10}$ .
- If a single file is used in read mode for a range of addresses, one line (hence, one datum) is read from the file and the file pointer is advanced to the next line for any address within the range. For example, if the example file is used for a range 0x2000-0x2004 and the read access is made to the addresses 0x2000, 0x2001, 0x2000, 0x2000, 0x2003 (in that order) during a simulation session, the values will go to the addresses in the order - 0x6666 to 0x2000, 0x9999 to 0x2001, 0xaaaa to 0x2000, 0xcccc to 0x2000 and 0x7f7f to 0x2003. Similarly, in write mode, when there is a write to any address in the range, one line containing that data is written in the file.
- If a Port Connect file is used in read mode with the no-rewind attribute, for any access made to the address after end-of-file is reached, the value 0xFFFF is read and the file pointer is kept unchanged. Otherwise the file pointer is rewound to the beginning and then one datum is read. For example, if the example file is used for address 0x2000 in read mode, at the time of the sixth read access to that address, the value 0xFFFF is read if no-rewind attribute is set. Otherwise 0x6666 is read. For information on how to use Port Connect features from the Command window, GEL files, or the Port Connect plug-in, please see the Code Composer Studio IDE online help.

## Peripherals

### Peripheral Interrupt Expansion (PIE)

PIE is typically used in systems where there are more interrupt sources than inputs into the core. This peripheral will have configuration capabilities to prioritize multiple interrupt sources into a finite set of interrupt inputs. The PIE basically feeds the interrupt vectors to the core if enabled. The PIE must only respond to vector fetches if  $VMAP = 1$ .

The PIE block serves to multiplex numerous interrupt sources into a smaller set of interrupt inputs. The PIE block can support 96 individual interrupts. The 96 interrupts are grouped into blocks of 8 and each group is fed into one of 12 core interrupt lines (INT1n to INT12n). Each of the 96 interrupts is, supported by its own vector stored in a dedicated RAM block that can be overwritten by the user. The vector is, automatically fetched by the CPU on servicing the interrupt. It takes eight CPU clock cycles to fetch the vector and save critical CPU registers. Hence the CPU can quickly respond to interrupt events. Each individual interrupt can be enabled or disabled within the PIE block.

For more details about the PIE, refer to the *TMS320F28x Control and Interrupts Reference Guide* (literature number SPRU078).

PIE module functions as per F2810 PIE specifications. The peripheral interrupts to PIE will be through pin files. The pin files can come from Timer0 or Timer1 dynamically. For more details about the connection of TIMER to PIE, see next section.

### Configuration File Syntax for PIE

```
module PIE;
    cssi_library <pie.dll>;           // dll path of PIE
    init_function InitPIE;           // PIE Initialization function
    int1         <file1>;            // interrupt pin file to intr 1 of PIE
    int2         <file2>;            // interrupt pin file to intr 2 of PIE
    int3         timer0 ;             // intr 3 is coming from timer0 output
    int4         timer1 ;            // intr 4 is coming from timer1 output
    ...
    ...

    int95        <file n-1>;         // interrupt pin file to intr 95 of PIE
    int96        <file n>;           // interrupt pin file to intr 96 of PIE
end PIE;
```

#### Note:

- Absolute path names CANNOT be specified for the Pie DLL path. The DLL path needs to be relative.
- The PIE vector table needs to be mapped in the simulator configuration file.

### Pin Connect File Format for PIE Interrupts

The files to store interrupts should have the cycle count at which interrupts had to be given to the PIE.

For example,

```
120 160 200
```

indicates that the interrupt is given to PIE at the 120th, 160th and 200th cycles.

# TMS320C28x INSTRUCTION SET SIMULATOR

## TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

---

### **Connection Protocol with TIMER**

As per F2810 specifications, Timer0 and Timer1 interrupts can be connected to PIE. This can be achieved by the providing the arguments timer0 or timer1 to the interrupts.

```
module PIE;
    cssi_library <pie.dll>;           // dll path of PIE
    init_function InitPIE;           // PIE Initialization function
    int3         timer0 ;             // intr 3 is coming from timer0 output
    int4         timer1 ;             // intr 4 is coming from timer1 output
end PIE;
```

**Note:** By default, the Timer is enabled.

### **Debugger visibility**

All the internal registers of PIE can be viewed from the debugger.

## VBUS

To enable migration of peripherals between various TI-DSP families of devices, the F2810/12 adopts the VBUS standard for peripheral interconnect. The VBUS Bridge multiplexes the various busses that make up the processor "Memory Bus" into a single bus consisting of 16 address lines and 16 or 32 data lines and associated control signals. There are two versions of the VBUS supported on the F2810/12. One version, VBUS16, only supports 16-bit accesses, retaining compatibility with C240x compatible peripherals. The other version, VBUS32, supports both 16- and 32-bit accesses and is used to connect newer peripherals requiring higher throughput.

For more details about the VBUS, see the *TMS320F28x Control and Interrupts Reference Guide* (literature number SPRU078).

VBUS16 and VBUS32 are compliant with the F2810 specifications. Users can use the file mechanism for supplying data to VBUS (VBUS reads) or recording (trace file) the VBUS data accesses (both read and writes). Users can have both VBUS16 and VBUS32 in the configuration file.

### Configuration File Syntax for VBUS

VBUS16 or VBUS32 are treated as memory blocks, and therefore need to be declared within the memory module sections. This section describes the syntax for specifying VBUS. A description of keywords is found in Table 7.

```
memory <module_name>;
    read_waitstates <wait_state_count>;
    write_waitstates <wait_state_count>;
    filein <input_file_path> ;
    tracefile <trace_file_path>;
    type <vbus_type>;
end <module_name>;
```

**Table 7. Keyword Description for Specifying VBUS in the Simulator Configuration File**

KEYWORD	ARGUMENT	DESCRIPTION
TYPE	<vbus_type> Can be one of the following: <ul style="list-style-type: none"> <li>● VBUS16</li> <li>● VBUS32</li> </ul>	Specifies the 16-bit (VBUS16) or 32-bit (VBUS32) versions of VBUS.
READ_WAITSTATES	<wait_state_count>	Specifies the number of read wait states. This value defaults to 0.
FILEIN	<input_file_path>	Specifies the input file for any reads from VBUS16 or VBUS32.
WRITE_WAITSTATES	<wait_state_count>	Specifies the number of write wait states. This value defaults to 0.
TRACEFILE	<trace_file_path>	Specifies the trace file for recording all VBUS activities. The file name will be suffixed with .vbus16 or .vbus32 for VBUS16 and VBUS32, respectively.

For example,

```
memory mem1;
    read_waitstates 8;
    write_waitstates 2;
    filein ..\tutorial\sim28xx\vbus\input32.txt ;
    tracefile ..\tutorial\sim28xx\vbus\trace32.txt;
    type VBUS32;
end mem1;
```

**Note:** Any other fields in the scope of VBUS configuration specifications will not be parsed and will have no effect. If FILEIN is not specified, reads will be 0s.

# TMS320C28x INSTRUCTION SET SIMULATOR TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

---

## **VBUS16 Input File Format**

<b>#Address</b>	<b>Data</b>
0x7000	0x3, 0x4, 0x5;
0x7001	0xff,0xfe,0xfd, 0x1,0x666,0xac,0x55;
0x7002	0x45, 0x34;
0x7003	78, 56,0x56;
0x7030	0x67, 0x45;

The first read from 7000 would access 0x3.

The second read from 7000 would access 0x4.

The third read from 7000 would access 0x5.

If the data points are over for any given address, any further access to this location, (7000) would wrap back to the first data point (i.e., the fourth access to 7000 would result in reading 0x3).

### **Notes:**

- Comments start with #
- Each address data pair should end with semi-colon.
- Address, data values can be either hex or decimal.
- All the data has to be 16 bits wide

For VBUS16, all the data must be 16 bits wide. Multiple data words can be attached to the same address. This would help to simulate the behavior of receiving data from a serial port to VBUS.



## VBUS16 Trace File Format

#-----#					
#CYCLE	TYPE	DS1	DS0	ADDRESS	DATA
#-----#					
16	READ	0	1	00007000	xxxxxxx
25	RDATA	x	x	xxxxxxx	00030003
30	READ	1	0	00007003	xxxxxxx
39	RDATA	x	x	xxxxxxx	004e004e
44	READ	0	1	00007000	xxxxxxx
53	RDATA	x	x	xxxxxxx	00040004
58	READ	1	0	00007003	xxxxxxx
67	RDATA	x	x	xxxxxxx	00380038
75	WRITE	0	1	00007004	ffffff
85	WRITE	1	0	00007007	fefefefe

### Notes:

- CYCLE indicates the CPU cycle counts.
- For each Vbus read, there will be two entries. One will have the entry for latched request and another will be for Vbus response. The difference of cycle numbers indicates the number of waitstates.
- TYPE can be READ, WRITE or RDATA.
  - READ: VBUS read request
  - WRITE: VBUS write request
  - RDATA: Valid data coming on VBUS
- DS1 corresponds to the odd address access
- DS0 corresponds to the even address access
- ADDRESS indicates the VBUS address. ADDRESS is valid for VBUS requests, and will be xxxxxx in the cycle, when the data is received to the VBUS.
- DATA will be xxxxxx for a READ request, but will be valid in the cycle, when VBUS actually gets the data.

## VBUS32 Input File Format

Input file format for VBUS32 is the same as that of VBUS16. For 16 bit accesses, it works the same way as VBUS16, for 32 bit accesses VBUS32 will read the 32 bit data.

**Note:** All data values in the input file must be 1 word (16 bits).

## VBUS32 Trace File Format

VBUS32 trace file format is the same as VBUS16, except that it can do 32 bit read/write access (where DS0 and DS1 are both set to '1').

# TMS320C28x INSTRUCTION SET SIMULATOR

## TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

---

### ***Pseudo Register Support VBUS Block Protection***

In order to support block enable at loading or initialization time, pseudo registers are provided, which can be set from the debugger command window.

These pseudo registers are as follows:

- ENPROT: Setting this bit to '1' will enable block protection, default is '0'.
- PROTSTART: 22-bit start address. The user needs to provide only the upper 16 bits.
- PROTRANGE: 22-bit address range. The user needs to provide only the upper 16 bits.

For example,

To enable block protection for the entire VBUS range, type the following commands in the command window:

? ENPROT=0x1

? PROTSTART=0x180

? PROTRANGE=0x80

### ***Debugger Visibility***

At any point in time, the data read by the simulator can be watched from the memory watch window. The first data for each address will be available for the simulator and will be displayed in the watch window. The memory window gets updated with the next value after each read from VBUS memory space.

## Timer

Timers are used extensively in HDD systems. Some timers are very specific and tied to hardware events, while other timers are tied to software events. The C28x core will ALWAYS be accompanied by three 16-bit timers. One timer is exclusively used for Real-Time OS (RTOS) applications, and the other two timers are for general purpose use.

Timers 0, 1 and 2 are identical 32-bit timers with pre-settable periods and with 16-bit clock pre-scaling. The timers have a 32-bit count down register, which generates an interrupt when the counter reaches zero. The counter is decremented at the core clock speed divided by the pre-scale value setting. When the counter reaches zero, it is automatically re-loaded with a 32-bit period value. Timer 2 is reserved for RTOS applications and is connected directly to INT14 of the CPU. Timers 0 and 1 are for general use. The Timer 0 interrupt can be connected to the PIE block.

For more details about the Timers, see the *TMS320F28x Control and Interrupts Reference Guide* (literature number SPRU078).

### Configuration File Syntax for Timer

```
module Timer;
    index 0/1/2;                // When index is 2, the timer connects to INT14
    cssi_library <timer.dll>;    // Path name of the Timer peripheral dll
    init_function initTimer;     // Initialization function
    pinc <pinc file> ;           // optional, generate the pinc files with cycle counts to a file
end Timer;
```

**Note:** Absolute path names cannot be specified for the timer dll path. The dll path needs to be relative. Index field is a must, without this field, Code Composer Studio IDE will give a configuration file error.

### Debugger Visibility

All the timer memory mapped registers can be viewed from the data memory windows. It requires simulator command file to have these address ranges mapped.

### Interrupt File Format

The files generated to store interrupts would have the cycle count at which they are generated. For example, the generated pinc file will have the contents,

```
120
160
200 // Indicates, interrupt is generated from timer at cycles 120, 160
```

# TMS320C28x INSTRUCTION SET SIMULATOR TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

---

## Flash

Flash memory is a non-volatile, electrically erasable and programmable memory. The Flash module may be configured as either program or data memory, or a combination. For more details about Flash memory, see the *TMS320F28x Control and Interrupts Reference Guide* (literature number SPRU078).

Flash functionality is compliant with F2810 Flash specification version 0.07.

### Features Implemented

- All FLASH registers are EALLOW protected.
- Read from Data and Program space will be supported.
- Pre-fetch mechanism is supported.
- Will have Random and Page and OTP wait states.
- Will support all three Power modes, "Sleep" "Standby" and "Active".
- Pipeline modes are supported.

### Features Not Supported

- Flash registers are NOT CSM protected.
- OTP, Flash bank will NOT be CSM protected.
- Program, ERASE, Test, Verify and Compaction features will NOT be supported.
- Does not support Flash Security Feature.

### Configuration File Syntax for Flash

```
Module main;
    cssi_modules Flash;
end main;

module Flash;
    cssi_library Flash.dll; // Flash DLL module
    init_function initFlash; // Init function
    first <start address1>;
    last <end address1>;

    first1 <start address2>;
    last1 <end address2>;

    first2 <start address3>;
    last2 <end address3>;
    .....
end Flash;
```

## **Configuration File Syntax for Flash (continued)**

### **Notes:**

- Absolute path names cannot be specified for the Flash dll path. The dll path needs to be relative. If the path is not specified, then it is assumed to be in the <CCS INSTALL DIR>\ti\drivers directory.
- By default, the specified memory will be shared in both program and data memory.
- The addresses will not be checked against any device specific mapping. This allows customizing the Flash module to any memory region.
- The FIRST, LAST, FIRST1, LAST1 order must be maintained. Every FIRST should have a corresponding LAST. Otherwise, Code Composer Studio IDE will give a configuration file error.
- The LAST address should be greater than "first". Otherwise, Code Composer Studio IDE will give configuration file error.
- WAITSTATES can not be specified, as the waitstates are configured through Flash configuration registers.
- The Flash memory (OTP and Flash memory) can be specified either in internal memory or in external Flash peripheral module. It should not be specified in both.

## **Debugger Visibility**

All the Flash memory mapped registers can be viewed from the data memory windows. This requires the simulator command file to have these address ranges mapped.

All Flash memory maps that are specified in the simulator configuration file can be viewed from the memory windows, either in Program or Data space. This requires the simulator command file to have these address ranges mapped.

# TMS320C28x INSTRUCTION SET SIMULATOR

## TECHNICAL OVERVIEW

SPRU608B – JULY 2002 – REVISED OCTOBER 2003

---

### Performance Numbers

Table 8 shows the performance numbers of the simulator. These numbers were gathered on a 900 MHz Intel™ Pentium™ III PC with 128MB of RAM. The application used for measurement is RSMBC (Reed-Solomon encoding and decoding).

**Table 8. Performance Numbers of the C28x Simulator**

SIMULATOR CONFIGURATION	SIMULATOR SPEED (IN KILOCYCLES/SECOND)
F28xx	24

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

---



## Related Documentation

***TMS320C28x DSP CPU and Instruction Set Reference Guide*** (literature number SPRU430) describes the central processing unit (CPU) and the assembly language instructions of the TMS320C28x™ fixed-point digital signal processors (DSPs). It also describes emulation features available on these DSPs.

***TMS320C28x Simulator User's Guide*** (literature number SPRU003) describes the basic capabilities of the simulator and the features provided for configuring it.

***TMS320C28x Code Composer Studio Tutorial*** (literature number SPRH138) is an online help tutorial available through Code Composer Studio Help menu.

***TMS320F28x DSP System Control and Interrupts Reference Guide*** (literature number SPRU078) includes information on the Flash and OTP memories, CSM, clocking mechanisms, GPIO registers, peripheral frames and peripheral interrupt expansion (PIE).

***TMS320F2810 Digital Signal Processor*** (literature number SPRS174) data sheet contains the pinout, signal descriptions, as well as electrical and timing specifications for the TMS320F2810 device.

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

### Products

Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>

### Applications

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

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