

TPU Debugger

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents	
ICD In-Circuit Debugger	
ICD Add-Ons	
TPU Debugger	1
TPU Basics	4
Entering TEST-Mode	5
TPU.BASE	Base address 5
TPU.SCAN	Scannig TPU 5
TPU.view	View TPU channels 6
TPU.Register.ALL	Register operation mode 6
TPU.Register.NEWSTEP	New debugging mode 7
TPU.Register.view	Register display 8
TPU.Register.Set	Register modification 9
TPU.Dump	Memory display 9
TPU.ListEntry	Table display 10
TPU.List	View microcode 11
TPU.Break	Break TPU 11
TPU.Go	Start TPU 12
TPU.SELect	Select TPU for debugging 12
TPU.Step	Single step TPU 13
TPU.RESet	Disable TPU debugger 13

TPU Basics

The TPU module is a complex free programmable RISC processor. TRACE32 supports single stepping as well as setting breakpoints in the RISC program. To start TPU debugging the emulation must be started in TEST mode. The following setup program example makes all preparations to debug the TPU on a MC68332 (the example can be found in `~/demo/m68k/etc/tpu`):

```
sys.res
sys.o test on
sys.m ai                                ; Important to enter TEST-MODE

d.s 0x0fffe12 %w 0x9999
d.s 0x0ffff00 %w 0x9d
d.s 0x0ffff04 %l 0x01000200
d.s 0x0ffff10 %w 0x9d
d.s 0x0ffff14 %l 0x00800100
d.s 0x0ffff20 %w 0x9d
d.s 0x0ffff24 %l 0x00200080
d.s 0x0ffff30 %w 0x9d
d.s 0x0ffff34 %l 0x00100030

d.s 0x0fffe1a %w 0x0aa
d.s 0x0fffe1e %w 0x0ea

tpu.b
w.tpu.r
w.tpu.view
w.tpu.l
enddo
```

After execution the screen shows the TPU registers, the RISC program and the TPU parameters. The command **TPU.Break** will stop the TPU and show the current values of the TPU registers. Stopping the TPU is only possible when the TPU is executing microcode, not when the TPU is in IDLE.

Entering TEST-Mode

To work with the TPU-Debugger the device must enter TEST mode. After starting up the emulation system, the **ETM** bit in the test submodule control register must be set (for CPU32 and CPU16 usually at \$YffA38). This register can only be written once. If the target program writes to this register, please change the program to set this bit correctly. The **ETM** bit is also set by the **TPU** debugger commands, but only if the target program has not already written to this register. If you want to debug your own microcode, you must also ensure that the TPU emulation RAM is configured by your target program. The TPU microcode cannot be modified by the debugger.

On the MPC555 (Rev. G and up) the test mode is enabled with the following command:

```
d.s 2fc3fc %long 80000000
```

TPU.BASE

Base address

Format:

TPU.BASE <address>

Selects the base address of the internal peripherals for the TPU debugger. This command is normally not required, as the base address is adjusted automatically according to the system settings.

See also

■ [TPU.view](#)

TPU.SCAN

Scannig TPU

Format:

TPU.SCAN

This command reads all TPU-registers and the microcode. This command is not required for normal usage. Entering this command should not change any TPU registers (except the timer registers will advance).

See also

■ [TPU.view](#)

Format:TPU.view [<file>]

Display the TPU configuration for each channel. The command is a predefined **PERipheral** window. The definition of this window can be changed either by entering another filename as an argument, or by modifying the file 'pertpu.t32' in the system directory. By this modification you can change the display to your special needs, e.g. add symbolic names for operation-modes and parameters. The TPU channels can also be viewed with the standard **PER** command in a more functional oriented manner.

CH	FUNC	PRI0	HSF	HSR	IEF	ISF	LNK	SGL	CHS	PRM0	PRM1	PRM2	PRM3	PRM4	PRM5	PRM6
0	\$9	mid	\$0	\$0	no	yes	no	no	no	009D	9710	0100	0200	9910	1414	
1	\$9	mid	\$0	\$0	no	yes	no	no	no	009D	9812	0080	0100	9912	0208	
2	\$9	mid	\$0	\$0	no	yes	no	no	no	009D	9813	0020	0080	9893	00C9	
3	\$9	high	\$0	\$0	no	yes	no	no	yes	009D	984F	0010	0030	984F	0002	
4	\$0	off	\$0	\$0	no	no	no	no	no	C345	E007	D90F	3421	8111	8080	
5	\$0	off	\$0	\$0	no	no	no	no	no	2200	3511	4320	629B	0951	2812	
6	\$0	off	\$0	\$0	no	no	no	no	no	8904	0348	D090	E20A	1542	2201	
7	\$0	off	\$0	\$0	no	no	no	no	no	320D	6000	021C	0F00	0101	8010	
8	\$0	off	\$0	\$0	no	no	no	no	no	2919	7634	4204	2810	3791	2C13	
9	\$0	off	\$0	\$0	no	no	no	no	no	3000	6001	1001	04A0	A112	0191	
10	\$0	off	\$0	\$0	no	no	no	no	no	6011	4361	0020	0240	3C01	2492	
11	\$0	off	\$0	\$0	no	no	no	no	no	0000	24C0	FFFF	FFC4	0000	0000	
12	\$0	off	\$0	\$0	no	no	no	no	no	0000	2540	0000	002F	FFFF	FFE4	
13	\$0	off	\$0	\$0	no	no	no	no	no	0000	0000	0000	2978	FFFF	FFC4	
14	\$0	off	\$0	\$0	no	no	no	no	no	0000	24AE	0000	0000	0000	1AEC	0000
15	\$0	off	\$0	\$0	no	no	no	no	no	0000	24DC	0000	0000	0000	24C0	0000

See also

- TPU.BASE
- TPU.Break
- TPU.Dump
- TPU.Go
- TPU.List
- TPU.ListEntry
- TPU.RESet
- TPU.SCAN
- TPU.SELect
- TPU.Step
- TPUBASE.ADDRESS()

TPU.Register.ALL

Register operation mode

Format:TPU.Register.ALL [ON | OFF]

In **ON** mode all registers of the TPU and the microcode are accessible. In TPU1 debugging mode you cannot step through a state-transition. This is the default mode. In **OFF** mode the debugger is limited to registers accessible direct by Scan-Path. This allows stepping through state-transitions.

See also

- TPU.Register.view

Format: **TPU.Register.NEWSTEP [ON | OFF]**

In **ON** mode the debugger is switched to TPU2 compatible mode (this is the default for the TPU2). Usually this command is not required.

See also

■ [TPU.Register.view](#)

Format:TPU.Register.view

The TPU-registers have two operation modes. The modes are selected with the command **TPU.Register.ALL**. In the limited mode only the registers available by Scan-Path are accessible. The full mode allows to access all registers of the TPU. The window contains also some registers not mentioned in the TPU-manual.

BCR'	This register contains the flags after executing one microinstruction. The information is used for conditional branches. This register is also used to display the flag-bits V, N, C, Z.
CHANNEL	The register is derived from the 'Decoded Channel Number Register' of the TPU (\$Yffe26). LINK is the 'Link Register' at \$Yffe22.
SGLR	'Service Grant Latch Register' (\$Yffe24).
DECOP	This register is a pseudo register, indicating the state of the decrement operation. The value 0 means no decrement active. Value 1 means that a repeat operation is running. A value of 2 means, that the decrementor is running and will execute a 'return' if it reaches zero. This register is only available (and required) when the TPU1 compatible debugging is used.
FLUSH	The pseudo-register is set to one, if the current instruction is not executed due to a 'flush' bit in the previous command.
TRANS	This register is a read-only register, that is set to one, if the TPU debugger has finished a state-transition. It is not possible to step through a state-transition in the full-debug mode. If the debugger detects that the next steps are doing such a transition, it executes enough steps to perform this transition and sets the pseudoregister TRANS to one.

```
UINST 0D452FFFF call 52; flsh.

V _ FL1 _ UPC    51  A      10  TCR1 18FE
N _ FL0 _ BCR   111  SR   981F  TCR2   0
C _ TDL _ BCR'  110  P    981F  MER   984F
Z _ MRL M HSCR 3ED0  DIOB 984F  ERT   984F
      LSL _ RAR    51              CHAN   3
      SEQ 0 DEC    0F  CHANNEL 3  TRANS   1
      PIN H DECOP   0  LINK  .....
      HSR 0 FLUSH   0  SGLR  .....
```

Format:TPU.Register.Set <regname> [<value>]

TPU.Register.Set

Register modification

Format:TPU.Register.Set <regname> [<value>]

The registers **BCR'**, **MER**, **CHANNEL**, **LINK** and **SGLR** cannot be modified.

TPU.Dump

Memory display

Format:TPU.Dump

Displays a hex-dump of the microcode.

0000	3FFFFFFE	BFFFFFF8	7FF9FEFA	11FDF80B
0004	8E12FEFF	1FFFFA03	3EFFF007	11FCF80F
0008	B20EFFFF	7FF9FEFF	8E0DFEFF	1FFFF203
000C	30FFD202	30FFE202	8E11FFFF	1FFFF203
0010	30FFD202	30FFE202	7FF9FEFE	11FDF80B
0014	8E12FFFF	1FFFFA03	D207FFFF	3EFFC007
0018	E1E401C7	8E1CFED0	7859FFFF	3A5FFFFF
001C	B22EFFFF	1FFFF00F	CFFF3013	3C7FF803

Format:	TPU.ListEntry
---------	---------------

Displays the entry-table of the microcode in symbolic format.

0180	161D	F00	1	X	X	0	X	001D	p := prm0; me
0180	E013	F00	1	X	X	1	X	0013	p := prm7
0181	8183	F00	2	X	X	X	X	0183	p := prm4
0181	F6FF	F00	3	X	X	X	X	00FF	p := prm7; me
0182	C166	F00	0	0	1	0	0	0166	p := prm6
0182	F003	F00	0	0	1	0	1	0003	p := prm7; me
0183	30FE	F00	0	0	1	1	0	00FE	p := prm1; me
0183	F013	F00	0	0	1	1	1	0013	p := prm7; me
0184	505D	F00	0	1	0	0	0	005D	p := prm2; me
0184	FFD2	F00	0	1	0	0	1	01D2	diob := prm7; me
0185	60F9	F00	0	1	0	1	0	00F9	p := prm3
0185	FED7	F00	0	1	0	1	1	00D7	diob := prm7; me
0186	BFFF	F00	0	1	1	0	0	01FF	diob := prm5; me
0186	FFC8	F00	0	1	1	0	1	01C8	diob := prm7; me
0187	3E7F	F00	0	1	1	1	0	007F	diob := prm1; me
0187	F80E	F00	0	1	1	1	1	000E	diob := prm7; me
0188	D38D	F01	1	X	X	0	X	018D	p := prm6; me
0188	FFFF	F01	1	X	X	1	X	01FF	diob := prm7; me
0189	3C7F	F01	2	X	X	X	X	007F	diob := prm1; me
0189	F80B	F01	3	X	X	X	X	000B	diob := prm7; me

See also

■ [TPU.view](#)

Format:

TPU.List

Displays the microcode. You can display either the microcode in ROM or the code in RAM. The microcode in RAM cannot be modified. The bar indicating the microprogram-counter is only displayed correctly, if a TPU-register window is on the screen too. The bar is displayed at the uPC position, i.e. at the address for the next instruction-fetch.

```
004D|7859FEFF ert := tcr1; neg_tdl;neg_mrl; neg_lsl.
004E|7A59FEFF ert := tcr2; neg_tdl;neg_mrl; neg_lsl.
004F|3C7FF807 diob := ert; ram diob -> prm1.
0050|D452FFFF call 52; flsh.
0051|525CB5FA ert := a + sr; pac := low; neg_mrl; write_mer; pir; end.
0052|163FF00B sr := diob; ramp <- prm2.
0053|101DF80F a := p, cc; ram diob<- prm3.
0054|8659FFFF if LOW_SAME then goto 59.
0055|36FEB013 p := diob + sr; ramp -> prm4.
```

See also

■

[TPU.view](#)

▲

['Release Information' in 'Legacy Release History'](#)

Format:

TPU.Break [<address>] [<channel>] [<flags>]

<channel>:

0..15.

<flags>:

BH | BL | BM | BT

There are three types of breakpoints: uPC breakpoints, channel breakpoints and state breakpoints. uPC and channel breakpoints can only be set when the TPU is stopped. The other breakpoints can also be set, when the TPU is running. The breakpoint-flags are described in detail in the TPU-Manual from Freescale Semiconductor. All breakpoint-types can be set in combination within one command. The command can also be used, when the TPU is already in IDLE-mode, to wait for execution.

```
tpu.b , , BT ; Break of TDL is asserted at beginning of state.
```

See also

■

[TPU.view](#)

Format:	TPU.Go [<address>] [<channel>] [<flags>]
<channel>:	0..15.
<flags>:	BH BL BM BT

Start TPU processor in real time. Additional breakpoint conditions may be set. The TPU may be stopped by **TPU.Break**.

```
tpu.g 0x100      ; Run, till uPC 100 is reached.  
tpu.g , 0x4      ; Run, till channel 4 is served.
```

See also

- [TPU.Step](#)
- [TPU.view](#)

TPU.SELect

Select TPU for debugging

Format:	TPU.SELect [A B]
---------	---------------------------

Selects the TPU unit in controllers with more than one TPU. The command must als be used when the base address of the peripherals was changed..

```
tpu.sel A      ; debug TPU unit A
```

See also

- [TPU.view](#)

Format: **TPU.Step**

Single step through microcode. Single-stepping is only possible, if the TPU is stopped and not in IDLE state.

```
tpu.s                    ; single step
```

- See also
-
- [TPU.Go](#) ■ [TPU.view](#)
 - ▲ 'Release Information' in 'Legacy Release History'

Format: **TPU.RESet**

Disables the TPU debugger.

- See also
-
- [TPU.view](#)