AN1480
APPLICATION NOTE

Handling Traps with the ST120 Core

By Béatrice BALAY

The first part of this document describes the ST120 Trap Servicing.The second part presents the implemented software to deal with a 'TRAP' event. Finally, the third part provides the user with a way to debug his code when a 'TRAP' occurs during the execution of a program.

CUSTOMER SUPPORT

For any questions concerning the ST120 DSP please contact our experienced staff of engineers at the following address: st100.support@st.com.

TABLE OF CONTENTS PAGE

1 - ST120 TRAP SERVICING

1.1 - Introduction

The primary goal of traps is to inform the user that a problem occurs during the execution of a program and requires to be fixed. Two kinds of Traps can be distinguished : hardware traps and software traps.

Software traps are generated by the instruction 'trap'. This instruction is available in GP16 and GP32 Instruction Modes.

Hardware traps occur when the execution of a program has caused some exceptional conditions which prevent the program from continuing its normal execution. Hardware traps are driven by the following hardware elements : the ST120 Core, the DMC (Data Memory Controller) and the PMC (Program Memory Controller). Hardware Traps are then subdivided into 2 categories: precise and non-precise traps.

Precise Hardware Trap: precise trapping implies that all the instructions prior to the trapped instruction have completed and that the trap instruction and **none** of its successors been have executed. The Core and the PMC produce precise traps.

Imprecise Hardware Trap: non-precise trapping implies that all the instructions prior to the trapped instruction have completed and that the trap instruction and **potentially some** of its successors may have been executed. By the nature of the ST120 pipeline, all data memory accesses are dealt with the 'bottom' of the pipeline and so errors associated with those accesses can not be precise. Therefore, the DMC produces non-precise traps.

1.2 - Trap Sequence Overview

The management of traps is performed though the use of 'exceptions vectors'. These vectors are located at the start of the memory address space. The trap vector address is obtained by ORing the vector address offset to 0x0000_0000 in case of a Hardware Trap, by ORing the vector address to 0x0000_0020 in case of a software Trap. The following paragraphs detail the Trap sequence for Hardware and Software Traps.

1.3 - Hardware Trap Sequence

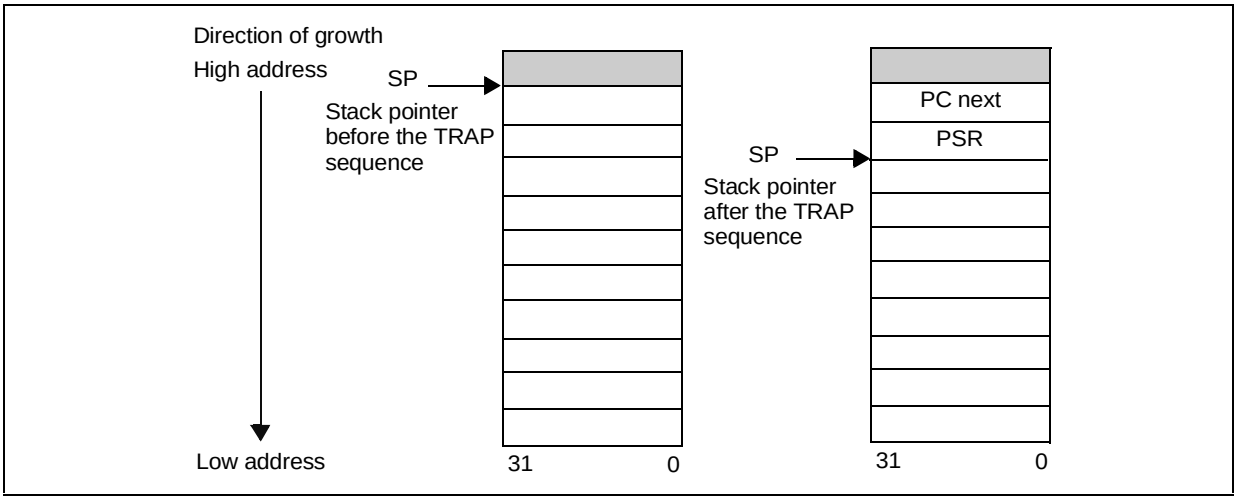
Each Hardware Trap has a corresponding Error Code Number [1].

For hardware Traps, a 6-bit bus EC[5..0] conveys the Error Code Number driven by the PMC, DMC or ST120 Core.

If the bit PSR.TE (Hardware Trap Enable) is set, a hardware trap causes the following sequence:

– **PCnext and the PSR register are saved on the stack** (P15 register, alias SP Stack Pointer). (See [1] Exceptions Handling -> Types of exceptions for more details on PC next):
PCnext -> [SP - 1]
PSR -> [SP - ? 2]

Figure 1 : Stack Overview Before and After a TRAP Sequence



– The Core switches to the following state :
  Supervisor Mode : PSR.PM <- 0b0
  Hardware Loops Disabled : PSR.LE <- 0b0
  GP32 Mode : PSR.IM <- 0b00
– The Core jumps to the following address :
  PC <- **0x0000_0000** + (EC << 2)

Notes: 1. If concurrent hardware traps occur simultaneously, the one having the highest error code is taken.
2. If hardware traps are disabled (PSR.TE cleared) and a TRAP condition occurs, the ST120 core performs the following :
   – *Core or a PMC trap condition* : the concerned instruction is executed as a "NOP" instruction and the program continues 'normally'.
   – *DMC trap condition caused by a Store operation* : the data write operation is not executed and the program continues 'normally'.
   – *DMC trap condition caused by a Load operation* : the data read operation is executed (with a potential false value) and the program continues 'normally'.
3. All cores and PMC TRAPs are taken independently of the guard status of the instruction.

1.4 - Software Trap Sequence

In GP32 mode, a Software Trap is executed as soon as "Gx? trap ImU4" instruction (ImU4 represents an immediate value of 4 bits) with "Gx" guard true is executed.

In GP16 mode, a Software Trap is executed as soon as "trap ImU4" is executed.

When a software trap is executed, the ST120 performs the following actions :

– **PCnext and the PSR register are saved on the stack** (P15 register, alias SP).
  (See [1] Exceptions Handling -> Types of exceptions for more details on PC next):
  PCnext -> [SP - 1]
  PSR -> [SP - ? 2]
– The Core switches to the following state :
  Supervisor Mode : PSR.PM <- 0b0
  Hardware Loops Disabled : PSR.LE <- 0b0
  GP32 Mode : PSR.IM <- 0b00
– The Core jumps to the following address :
  PC <- **0x0000_0020** + (Imu4 << 2)

This sequence is the same as the one for hardware traps except that the vector address computation is not the same. The base address to compute the vector address of a hardware trap is the reset address : 0x0000_0000 whereas the base address to compute the vector address of a software trap is : 0x0000_0020.

1.5 - Trap Vector Address

64 TRAPs are available. The purpose of the following table is to show the vector address of each trap. The TRAP description is detailed in [1] Exceptions Handlings -> Hardware Traps.

Table 1 : Trap Vector Address

Trap Type	Error Code	ImU4 Value	TRAP Description	Vector Address
8 Core TRAPS	0x 0	Not Applicable	Illegal (Reserved for Reset)	0x0000_0000
	0x 1 to 0x 4	Not Applicable	Reserved	0x0000_0004 -> 0x0000_0010
	0x 5	Not Applicable	TRAP.MISALIGN	0x0000_0014
	0x 6	Not Applicable	TRAP.PROTECT	0x0000_0018
	0x 7	Not Applicable	TRAP.OPCODE	0x0000_001c
16 Software Traps	n.a	ImU4 = [0 ..15]	TRAP ImU4	0x0000_0020 -> 0x0000_005c
8 PMC Traps	0x 18 -> 0x 1d	Not Applicable	Reserved	0x0000_0060 -> 0x0000_0074
	0x 1e	Not Applicable	Reserved	0x0000_0078
	0x 1f	Not Applicable	TRAP.POUTOFMEM	0x0000_007c
8 DMC Traps	0x 20	Not Applicable	TRAP.DOVERMEM	0x0000_0080
	0x 21	Not Applicable	TRAP.DSYSERR	0x0000_0084
	0x 22	Not Applicable	TRAP.DSYSMISALIGN	0x0000_0088
	0x 23	Not Applicable	Reserved	0x0000_008c
	0x 24	Not Applicable	TRAP.DMISALIGN	0x0000_0090
	0x 25	Not Applicable	TRAP.DOUTOFMEM	0x0000_0094
	0x 26	Not Applicable	TRAP.DVOLATILE	0x0000_0098
	0x 27	Not Applicable	Reserved	0x0000_009c
24 Hardware Traps	0x 28 -> 0x 3f	Not Applicable	Undefined	0x0000_00a0 -> 0x0000_00fc

Notes: 1. Hardware Traps can be globally enabled or disabled by clearing or setting the bit TE (Bit 5) of the PSR (Processor Status Register). Note that this bit has no effect on software Traps. This bit can be set or cleared by the software instruction 'BFPSR0 mask value'.

To enable Hardware Traps, write : `bfpsr0 0x10, 0x10`.

To disable Hardware Traps, write : `bfpsr0 0x10, 0x0`.

Since this instruction modifies the byte 0 in PSR, the core must be in SuperVisor Mode to execute those instructions.

After a Reset Operation, the Core is in Supervisor Mode and hardware Traps are disabled (See [1] Exception Handling -> Reset)

2. The ST120 Core allows the 'system' to select whether data memory errors will generate traps or not. By default, the ST120 Core does not generate Traps for data memory errors. The DMC imprecise Traps are detected by the diagnostic hardware (On Chip Emulator) that becomes responsible to inform the 'host' of data memory errors.

2 - SOFTWARE IMPLEMENTATION FOR TRAP HANDLERS

2.1 - The Trap Vector Table

The Trap vector table is the code placed in the program memory space reserved for trap vectors. As explained in the previous paragraphs, this program memory space ranges from 0x0000_0000 to 0x0000_00FC. Therefore, the first 256 bytes of the memory space are reserved.

Once PC has branched in the Trap vector Table, it then branches to a specific trap handler that is responsible for providing the user all the information to debug his code. Therefore, in the Trap Vector Table, we find 64 instructions 'goto associated_trap_handler'. TRAP handler can be user-defined for each TRAP.

The code of the Trap Vector Table is provided in a file called **<GHS_DIR>/libsrc/tvtable.s**. This file provides default trap handler names.

Hereafter is an extract of this file:

```
// Processor : ST100
// This file defines the trap handlers
// Default vectors jump to a label
.macro VECTOR label
goto %rel2to26(label)
.endm
// at 0x04 Core traps
VECTOR _hwtrap1   // code to place at  0x0000_0000 + 1 <<2
VECTOR _hwtrap2   // code to place at  0x0000_0000 + 2 <<2
VECTOR _hwtrap3   // code to place at  0x0000_0000 + 3 <<2
VECTOR _hwtrap4   // ....
VECTOR _hwtrap5
VECTOR _hwtrap6
VECTOR _hwtrap7
// at 0x20 Software traps
VECTOR _swtrap0
// ...
```

The Vector Trap Table is allocated to the .startup section that one will always find in a typical linker file. This program section must be placed to the address 0x0000_0000.

The first part of the code given hereafter shows the inclusion of the tvtable.s file in the startup section. This part of code is extracted from the file **<GHS_DIR>/libsrc/crt0.st1** (this file defines the minimum context for an application to be run) ([2] Chapter 21 Source Files Available For Customization).

```
#include "indsyscl.h"
        .file "crt0.st1"
        .section ".startup", .text
_start::
// Reset vector jumps to _start2
goto %rel2to26(_start2)              // Reset Address 0x0000_0000
// Inclusion of other trap handlers   from Address 0x0000_0004
        #include "tvtable.s"
        .type _start,@function
        .size _start,.-_start
```

This second part of code is an extract of a typical linker file where we see the memory placement of the .startup section.

```
# Default link script for the ST100 Core Implementation
# Memory mapping
MEMORY {
 program : org = 0x000000, len = 128K
 xspace  : org = 0x100000, len = 16K
 yspace  : org = 0x200000, len = 16K
 intios  : org = 0x300000, len = 1M
 extios  : org = 0x400000, len = 1M
 extern  : org = 0x500000, len = 11M
}
# Objects placement
SECTIONS {
    .startup 0x0  : > program
    .picbase 0x100: > program
    .thandlers    : > program
```

Note: 1. It is strongly recommended to use the provided names for each trap handler (_hwtrap1, ..._swtrap0,...). Otherwise, you will have to modify the file tvtable.s, build the new crt0.o before being able to link your application. The default file crt0.o is provided in <GHS_DIR>/st1_gp32 directory.

2.2 - Trap Handlers

A minimum trap handler is :

```
rte  // Return from Exception
```

A standard trap handler is :

```
// Operation Sequence
poprte registers_list   // Remove the context used by the Operation
                        // Sequence and Return from Exception.
```

The operation sequence is responsible for providing the programmer with the maximum diagnostic information in order to identify the cause of the present TRAP.

The 'poprte' instruction is described in [5].

2.3 - Default Trap Handlers

Green Hills Toolchain provides you a default TRAP handler for each TRAP. This TRAP handler is defined in <GHS_DIR>/libsrc/thandlers.s.

thandlers.s consists of an assembly macro that defines each TRAP handler. The default handler is the macro STOP_HANDLER you can use when the code is executed in debug mode i.e. with the 'diagnostic hardware' enabled. It prevents the code from keeping running by executing a 'bkp' instruction. To have more details on the 'bkp' instruction, refer to [5].

thandlers.s also provides an assembly macro DO_NOTHING_HANDLER that just returns to the instruction following the trapped one (for precise traps only) without providing any information.

Hereafter is provided the code of the thandlers.s file.

```
.section ".thandlers", .text
.define SYSCALL_EXIT 2
.macro HANDLER label
.weak label
.export label
label:
.endm
.macro STOP_HANDLER
barrier
bkp
.endm
// This handler can be used if one wants to ignore
// some traps
.macro DO_NOTHING_HANDLER
rte
.endm
.entry32
HANDLER _hwtrap1
HANDLER _hwtrap2
HANDLER _hwtrap3
......
HANDLER _hwtrap2F
STOP_HANDLER
```

Notes:
1. The file thandlers.s is placed in the program section .thanders.
2. The default TRAP handler routine is the same for each TRAP. It prevents the code from keeping running. It does not provide any information concerning the identification of the trap (hardware, software, error code) and the PC address at which one the trap occurs.

3 - HOW TO GET DETAILED TRAP HANDLERS

The following part presents a way to:

– Identify which trap has occurred (software, hardware, error code)

– Know at which PC address the trap has occurred.

3.1 - Data related to a TRAP

The information related to a TRAP can be stored in a C structure :

```
enum trap {soft,hard,undefined};
typedef struct
{ unsigned int address;
  int number;
 /*type=1 for HARDWARE TRAP, type=0 for SOFTWARE TRAP*/
  enum trap type;
} TRAP_st;
```

Do not forget to initialize the structure.

```
TRAP_st trap_st = {0,-1,undefined};
```

3.2 - The TRAP handler routines

To have detailed information about a TRAP using the previously presented structure, the TRAP handler has to :

– Initialize as required the C structure trap_st so that when a TRAP occurs, the user is able to identify it by reading the content of the structure trap_st.

Each Trap handler can be defined though the use of an assembly macro. We have called it TRAP_HANDLER. Its parameters are the label of the TRAP handler (defined in tvtable.s), its associated error code and the nature of the TRAP (software, hardware).

Hereafter is an extract of the file where one will define each TRAP handler :

```
.section ".thandlers", .text
.macro TRAP_HANDLER label, nb, type
.weak label
.export label
label:  push p10-p11,r0-r1
makeba p10,trap_st // Initialization of the structure  trap_st
make r0,nb
sdw @(P10+4),r0
ldw r1,@(SP+28)
sub r1,r1,4
sdw @(P10+0),r1
make r1,type
sdw @(p10+8),r1
loopena
call trap_print
poprte p10-p11,r0-r1
.endm
.entry32
/* Definition of the Core Hardware Traps */
TRAP_HANDLER _hwtrap1,1,1
TRAP_HANDLER _hwtrap2,2,1
.....
TRAP_HANDLER _hwtrap2F,64,1
/* Definition of the default trap_print function */
```

```
.text
.entry32
.globl trap_print
.weak trap_print
trap_print :
barrier
bkp
 rts
```

Notes:
1. A call to a user's function is made at the end of each macro to allow a customization of each Trap handler. In the given code, this function has been called trap_print. A default definition is also given which just prevents the code from keeping running. This function has been defined with a 'weak' label so that the user may define it again with his own code.
2. Inside the macro TRAP_HANDLER, all the Traps Handlers are defined as 'weak' labels so that the user may define again his own trap handler.
3. Trap Handlers can be defined at C level. Here is an example for the Software Trap 0:

```
__interrupt _swtrap0(void)
{ printf("A software Trap 0 has occured\n");
}
```

3.3 - A way to use the function trap_print

To allow a customization of the TRAP routines, a call to a function called trap_print is made inside each TRAP handler after the initialization of the C structure. By default, this trap_print function just performs a 'bkp' instruction that prevents the code from keeping running. When stopped in the trap_print function, the user can have a look at the trap_st structure to identify the trap.

One may want to print in the IO window of the host the value of each data in the structure. As the function trap_print has been defined with a 'weak' label, we can define it again. Hereafter is an example of a new definition of the function trap_print at C level :

```
#ifdef TRAP_PRINT
void trap_print(void)
{
    int address;
    int number;
    int type;
    address= trap_st.address;
    number=trap_st.number;
    type=trap_st.type;
#ifndef TRAP_PRINT_MAP
if (type == 0)
printf("A software Trap of number %d occurs at address
0x%8x\n",number,address);
if (type == 1)
printf("A hardware Trap of number %d occurs at address
0x%8x\n",number,address);
#else
/* A Printf can be specified for user/specific messages according the value of
the Trap Number */
if (type ==0)
{  printf("Software Trap number %d at address 0x%8x\n",number,address);
}
if (type ==1)
```

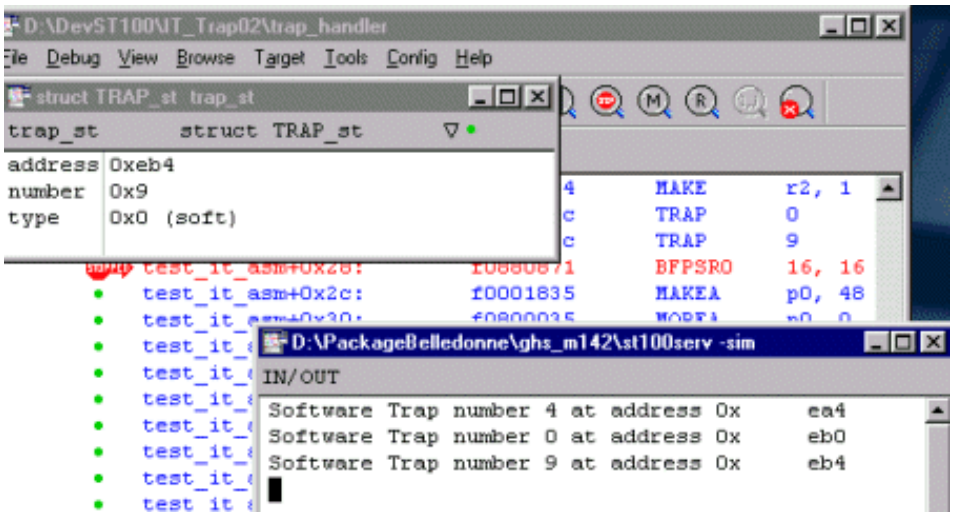
```
{ if (number < 5)
      printf("Hardware Trap Reserved from Core at address 0x%8x\n",address);
  if (number == 5)
      printf("Hardware Trap Misalign from Core at address 0x%8x\n",address);
  if (number ==6)
      printf("Hardware Trap Protect from Core at address 0x%8x\n",address);
  if (number ==7)
      printf("Hardware Trap Opcode from Core at address 0x%8x\n",address);
  if ((number >=24)&&(number <30))
      printf("Hardware Trap Reserved from PMC at address 0x%8x\n",address);
  if (number == 30)
      printf("Hardware Trap Excecute from PMC at address 0x%8x\n",address);
  if (number == 31)
     printf("Hardware Trap Out of Memory from PMC at address 0x%8x\n",address);
  if (number >=40)
      printf("Hardware Trap Undefined, Error Code = %d at address
0x%8x\n",number,address);
}
#endif
}
#endif
```

By defining the macro-constant **TRAP_PRINT**, the data related to the current trap are displayed in the IO window.

By defining the macro-constants **TRAP_PRINT** and **TRAP_PRINT_MAP**, the data related to the current trap plus its meaning (if defined) according to the Trap Mapping are displayed in the IO window.

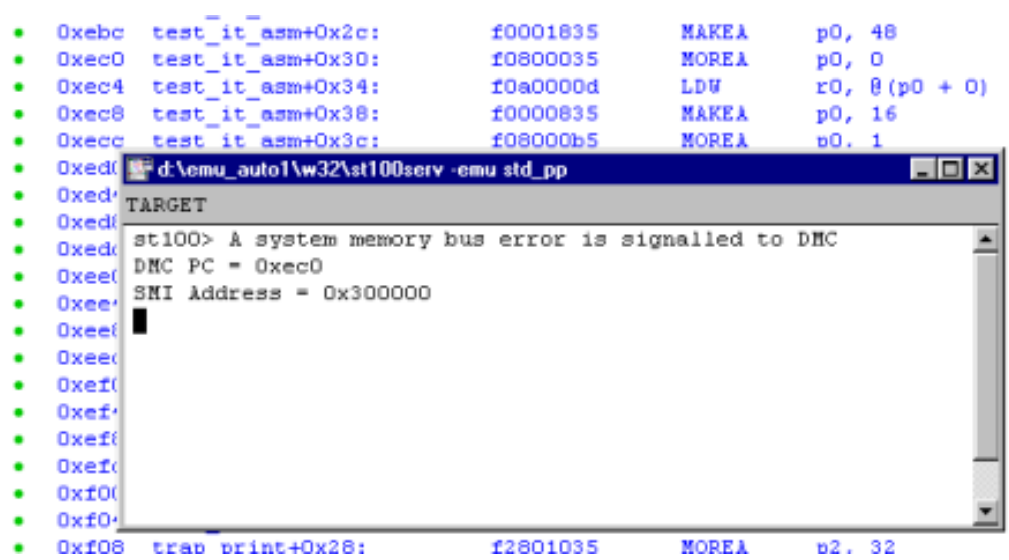
Hereafter (see Figure 2) is an example of the display of the content of the structure trap_st (See Browse -> Globals ->trap_st) after a software trap 9. It also shows the result of the trap_print function in the IO window.

Figure 2 : Display of the Content of the Structure trap_st



Notes:
1. The several level of "defines" (no #Define, #ifdef TRAP_PRINT) allow you to control the level of intrusion (library stdio.h, ...) of the detailed TRAP handlers in your application.
2. The TRAP handlers previously presented are not re-entrant. The content of the structure trap_st is the one corresponding to the latest trap.
3. **About imprecise traps :** only precise TRAPs can be traced with this software. Remember that only Core and PMC TRAPs are precise. Imprecise TRAPs driven by the DMC are generated some time after the error has occurred thus the architectural state of the machine will be UNDEFINED when entering in the TRAP handler. To work-around this, the ST120 provides a diagnostic hardware that contains the necessary mechanisms for precisely locating data memory access errors. Those information are sent to the host so the user has at his disposal all the required data to debug his code. Hereafter is an example of information (see Figure 3) returned by the diagnostic hardware (On Chip Emulator) to the host (GHS debugger):

Figure 3 : Example of Message Returned by the One Chip Emulator



REFERENCES

[1] ST120 DSP-MCU Core Reference Guide Release 1.3 - *STMicroelectronics*

[2] Embedded ST100 Development Guide v2.0 - Green Hills - Multi 2000 Release

[3] ST120 DSP-MCU Instruction Set Reference Guide Release 1.1 - *STMicroelectronics*

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.