ASHLING
EMBEDDED TECHNOLOGY AND INNOVATION

Ashling Product Brief APB197

Debugging eTPU application code using the Ashling MPC5500 tools

Contents

1 References

1. Freescale Enhanced Time Processing Unit (eTPU) Reference Manual: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=eTPU
2. Byte Craft eTPU_C Compiler: http://www.bytecraft.com/etpuccaps.html

2 Introduction

The Freescale MPC5500 Embedded Microprocessors consist of a PowerPC core (e200) and two Enhanced Time Processing Units (eTPUA and eTPUB). The eTPUs are intelligent, semi-autonomous co-processors designed for timing control, I/O handling, serial communications, motor control and engine control. The eTPUs have a unique instruction set, which is supported by a C compiler (created by Byte Craft and available from Ashling, see Ref. 2). This application note describes eTPU code development and debugging using:

- Byte Craft's eTPU_C Compiler
- Ashling's PathFinder for MPC5500 Source Debugger
- Ashling's Vitra for MPC5500 Emulator will Real-Time Trace

The application note uses the `PFMPC\Examples\eTPU` example program supplied with PathFinder-MPC5500 at ver. 103 and later.

3 Building your application

This section describes how to build the eTPU example `PFMPC\Examples\eTPU` using the Byte Craft tools. Skip ahead to the next section if you are already familiar with building eTPU applications.

The eTPU example is supplied fully compiled and linked in two versions:

1. `PFMPC\Examples\eTPU\SRAM` which runs from internal SRAM at 0x4000-0000
2. `PFMPC\Examples\eTPU\flash` which runs from internal Flash at 0x0000-0000

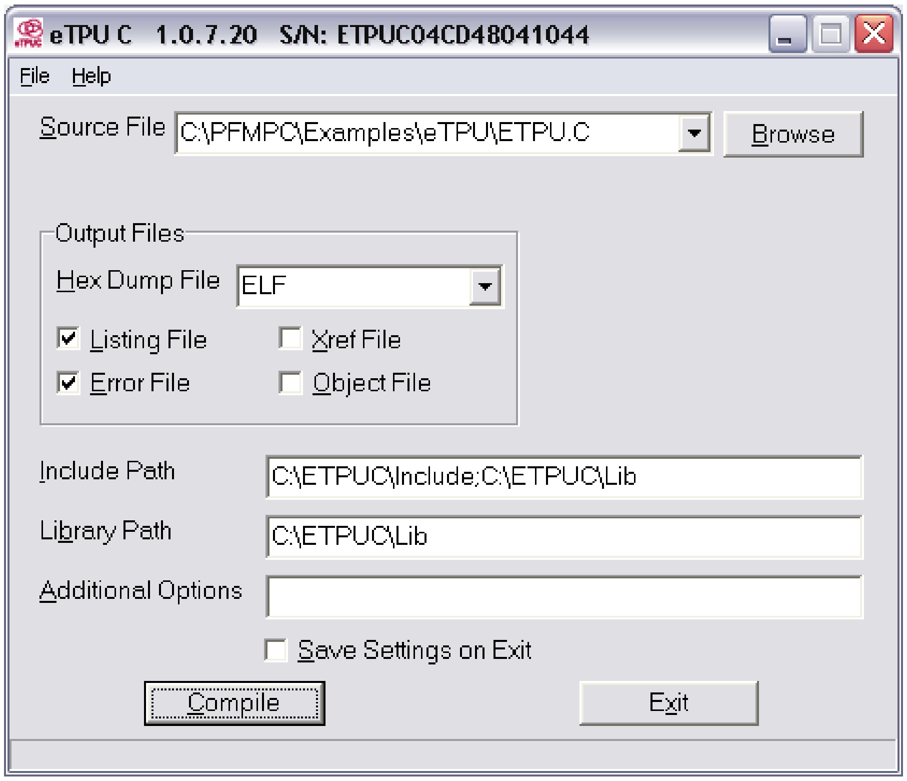
The Byte Craft compiler compiles the eTPU code to an ELF file (binary file containing program image and debug information) and a C-format Header file that contains the eTPU program image. The ELF file is necessary for eTPU source-debugging support in PathFinder.

The example program consists of two modules:

1. `e200.C` e200 application code written in C, compiled using the Metrowerks MPC5500 tools.
2. `eTPU.C` eTPU application code written in C, compiled using the Byte Craft eTPU tools. The Byte Craft compiler actually generates a header file (`eTPUCODE.H`) which contains all of the eTPU code, defined in a C array. This array is then included in e200.C and copied to eTPU memory (SCRAM) at run-time.

3.1 Compiling `eTPU.C`

To compile `eTPU.C`, we use the Byte Craft tools as follows:

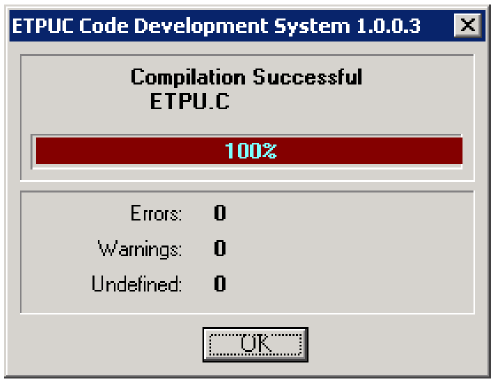


Source File: C:\PFMPC\Examples\eTPU\ETPU.C
Hex Dump File: ELF
[✓] Listing File
[✓] Error File
Include Path: C:\ETPUC\Include;C:\ETPUC\Lib
Library Path: C:\ETPUC\Lib
Additional Options:
[ ] Save Settings on Exit

Figure 1 Using the Byte Craft Compiler to compile `eTPU.C`

In the **Output Files** group ensure that you select **ELF** as the **Hex Dump File**. This instructs the Byte Craft compiler to generate an ELF output file. PathFinder uses the information within the generated ELF file for eTPU source-level debug. By default the name of the ELF file produced by the Byte Craft compiler is the name of the source file with a .ELF extension. (For this example the output file generated will be eTPU.ELF)

Click on **Compile** and the Byte Craft compiler will run and generate `eTPUCODE.H and eTPU.ELF.`



Compilation Successful
ETPU.C
100%

Errors: 0
Warnings: 0
Undefined: 0

Figure 2 Byte Craft Compiler in action

Generation of eTPUCODE.H was controlled by pragma definitions in eTPU.C as shown in Figure 3 below:

```
/* Information exported to Host CPU program */
#pragma write h, (::ETPUfilename (etpucode.h));
#pragma write h, (::ETPUliteral(#define GPIO_FUNCTION_NUMBER) 0                  );
#pragma write h, ( );
#pragma write h, (::ETPUliteral(#define GPIO_OUTPUT_INIT    ) GPIO_OUTPUT_INIT   );
#pragma write h, (::ETPUliteral(#define GPIO_OUTPUT_HIGH    ) GPIO_OUTPUT_HIGH   );
#pragma write h, (::ETPUliteral(#define GPIO_OUTPUT_LOW     ) GPIO_OUTPUT_LOW    );
#pragma write h, (::ETPUliteral(#define GPIO_OUTPUT_PULSE   ) GPIO_OUTPUT_PULSE );
#pragma write h, (::ETPUliteral(#define GPIO_OUTPUT_PULSES  ) GPIO_OUTPUT_PULSES);
#pragma write h, (::ETPUliteral(#define GPIO_OUTPUT_PROCA   ) GPIO_OUTPUT_PROCA );
#pragma write h, (::ETPUliteral(#define GPIO_OUTPUT_PROCB   ) GPIO_OUTPUT_PROCB );
#pragma write h, ( );
#pragma write h, (extern const char pucETPUCode[] = { ::ETPUcode };);
#pragma write h, ( );
```

Figure 3 #pragma definitions in eTPU.C

The generated `eTPUCODE.H` consists of some `#defines` (which were defined/used in `eTPU.C` and will be needed in `e200.C`) and the array `pucETPUCode[]` which contains the actual eTPU code, as in Figure 4 below:

```
#define GPIO_FUNCTION_NUMBER 0

#define GPIO_OUTPUT_INIT     1
#define GPIO_OUTPUT_HIGH     2
#define GPIO_OUTPUT_LOW      3
#define GPIO_OUTPUT_PULSE    4
#define GPIO_OUTPUT_PULSES   5
#define GPIO_OUTPUT_PROCA    6
#define GPIO_OUTPUT_PROCB    7

extern const char pucETPUCode[] = { 0x40,0x80,0x40,0x80,0x40,0x80,0x40,0x80,
                                    0x40,0x83,0x40,0x84,0x40,0x85,0x40,0x87,
                                    0x40,0x90,0x40,0x92,0xC0,0x94,0xC0,0x94,
                                    0xC0,0x94,0xC0,0x94,0xC0,0x94,0xC0,0x94,
                                    0xC0,0x94,0xC0,0x94,0xC0,0x94,0xC0,0x94,
                                    0xC0,0x94,0xC0,0x94,0xC0,0x94,0xC0,0x94,
                                    0xC0,0x94,0xC0,0x94,0xC0,0x94,0xC0,0x94,
                                    0xC0,0x94,0xC0,0x94,0xC0,0x94,0xC0,0x94,
                                             ....
                                             ....
                                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
```

Figure 4 Automatically generated `eTPUCODE.H`

The `eTPUCODE.H` must next be included in the main `e200.C` program; for example, by adding a #include, as in Figure 5 below:

```
/*****************************************************************************
       Module: E200.C
     Engineer: Ashling
  Description: This file contains the code which runs on the e200 core
Date           Initials    Description
12-May-2005    ASH         Initial
*****************************************************************************/
#include "mpc5554.h"
#include "etpucode.h"

#define ETPU_SCRAM      0xC3FD0000
#define ETPU_PRAM       0xC3FC8000
```

Figure 5 #Including the eTPU code in `e200.C`

At run-time, the eTPU code must be loaded into eTPU memory-space by the e200 core; for example, by using a `memcpy` call to the `pucETPUCode[]` array as in Figure 6 below:

```
/*****************************************************************************
     Function: InitialiseETPU
     Engineer: Ashling
        Input: none
       Output: none
  Description: initialises eTPUs, copies microcode to SCM, etc.
Date           Initials    Description
12-May-2005    ASH         Initial
*****************************************************************************/
void InitialiseETPU(void)
{
   // Set ETPU Entry Table Base Address
   ETPU.ECR_A.B.ETB = 0x0;
   ETPU.ECR_B.B.ETB = 0x0;
   // Stop both eTPU engines
   ETPU.ECR_A.B.MDIS = 1;
   ETPU.ECR_B.B.MDIS = 1;
   // Enable writing to SCM
   ETPU.MCR.B.VIS = 1;

   // Copy microcode into SCRAM
   (void)memcpy((char *)ETPU_SCRAM, pucETPUCode, 0x3000);

   // Disable writing to SCM
   ETPU.MCR.B.VIS = 0;
   // Restart both eTPU engines
   ETPU.ECR_A.B.MDIS = 0;
   ETPU.ECR_B.B.MDIS = 0;
   // Configure Time Base.
   ETPU.TBCR_A.B.TCR1CTL = 2;
   ETPU.TBCR_A.B.TCR1P   = 2;
   ETPU.TBCR_B.B.TCR1CTL = 2;
   ETPU.TBCR_B.B.TCR1P   = 2;
   // Configure Channel
   ETPU.CHAN[ETPUA_CHANNEL].CR.B.CPBA = ETPU_PRAM_OFFSET;
   ETPU.CHAN[ETPUA_CHANNEL].CR.B.CFS  = GPIO_FUNCTION_NUMBER;
   ETPU.CHAN[ETPUB_CHANNEL].CR.B.CPBA = ETPU_PRAM_OFFSET;
   ETPU.CHAN[ETPUB_CHANNEL].CR.B.CFS  = GPIO_FUNCTION_NUMBER;
   // Set Channel Priority
```

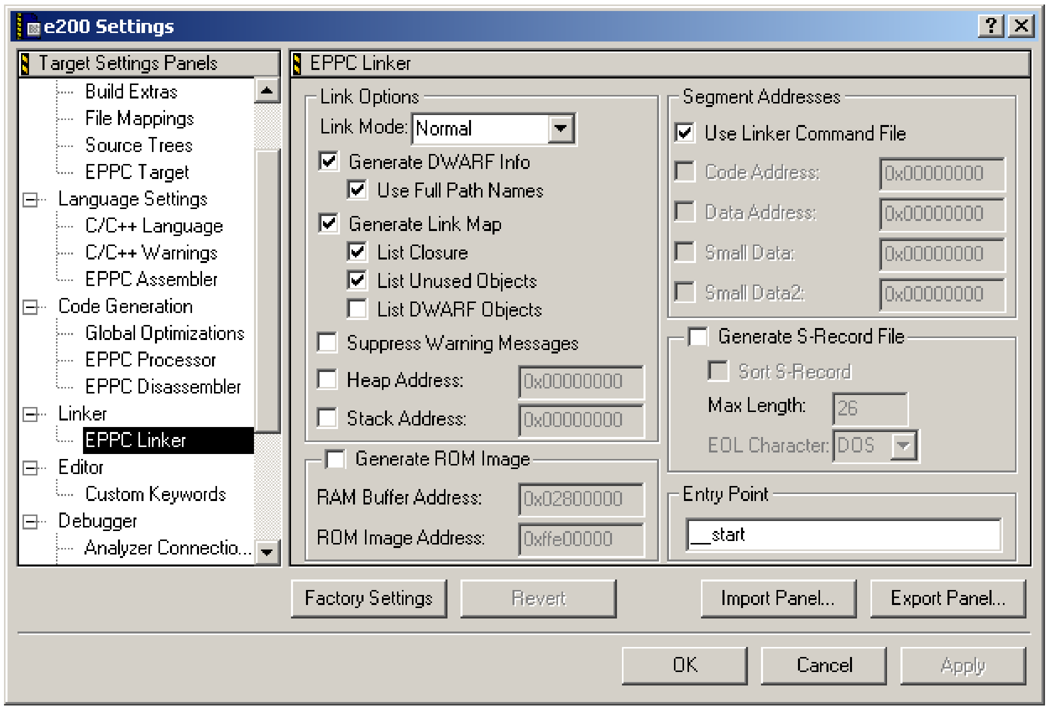
```
    ETPU.CHAN[ETPUA_CHANNEL].CR.B.CPR = 2;
    ETPU.CHAN[ETPUB_CHANNEL].CR.B.CPR = 2;
}
```

Figure 6 Copying the eTPU code into eTPU SRAM

3.2 Building `e200.C`

Next, the `e200.C` is built and linked using a batch-file, make-file or an IDE. Ensure that your MPC5500 Linker is configured to generate DWARF-format debugging information for source debugging.

For example, if you are using the Metrowerks MPC5500 tools you can load the project file `e200.MCP`, supplied with the Ashling MPC5500 tools, into CodeWarrior and you can build the program using CodeWarrior's **Project|Make**. To enable source-level debug we have set **Generate Dwarf Info** to true in the CodeWarrior e200 Linker Settings dialog as in Figure 7 below:



EPPC Linker

Link Options
Link Mode: Normal
☑ Generate DWARF Info
☑ Use Full Path Names
☑ Generate Link Map
☑ List Closure
☑ List Unused Objects
☐ List DWARF Objects
☐ Suppress Warning Messages
☐ Heap Address: 0x00000000
☐ Stack Address: 0x00000000

☐ Generate ROM Image
RAM Buffer Address: 0x02800000
ROM Image Address: 0xffe00000

Segment Addresses
☑ Use Linker Command File
☐ Code Address: 0x00000000
☐ Data Address: 0x00000000
☐ Small Data: 0x00000000
☐ Small Data2: 0x00000000

☐ Generate S-Record File
☐ Sort S-Record
Max Length: 26
EOL Character: DOS

Entry Point
__start

Figure 7 CodeWarrior Linker Options

3.3 Symfinder

After building both e200.C and eTPU.C, there should now be two resulting ELF files (e200.ELF and eTPU.ELF). You should now run the SymFinder for DWARF utility to generate a e200.CSO file which may be directly loaded into PathFinder. Both the e200.ELF file and the eTPU.ELF file (preceded by the ETPU switch) are passed to the SymFinder utility as follows:

```
> SFDWARF e200.ELF ETPU ETPU.ELF METROMPC
```

The `METROMPC` switch specifies that the file was generated by the Metrowerks compiler; for the GNU GCC MPC5500 Compiler/Linker you should use the `GNUMPC` switch as follows:

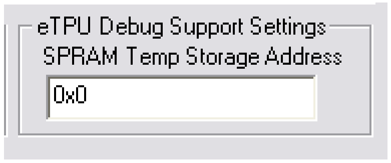
```
> SFDWARF e200.ELF ETPU ETPU.ELF GNUMPC
```

4 Debugging your eTPU application

4.1 Configuring PathFinder

4.1.1 Specifying SPRAM Temporary Storage

PathFinder needs four bytes of temporary storage in the Shared Parameter Memory area (SPRAM) for saving and restoring the eTPU registers during a read. This is specified in the **Configuration|MPC5500 Settings** dialog using the **SPRAM Temp Storage Address** as in Figure 8 below:

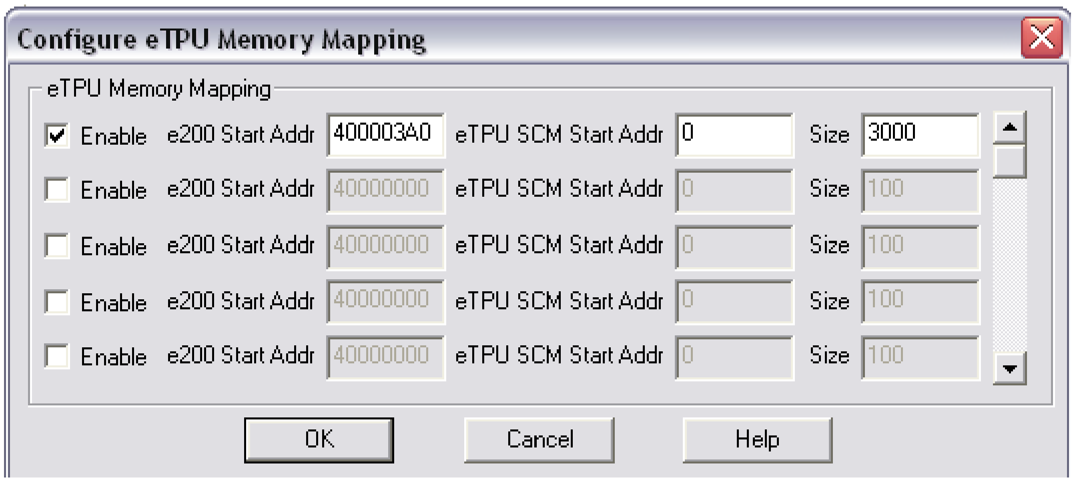


eTPU Debug Support Settings
SPRAM Temp Storage Address: 0x0

Figure 8 Specifying SPRAM Temporary Storage

4.1.2 eTPU Memory Mapping

The eTPU Memory Mapping dialog (**Configure|Configure eTPU Memory Mapping**) shown in Figure 9 below allows you to tell PathFinder where your eTPU code resides (from the perspective of the main e200 MPC55xx core). PathFinder needs this information to allow it to fill the **Disassembly (eTPU)** window and to reconstruct eTPU trace information (that is, to show the actual eTPU instructions as traced and executed) because PathFinder cannot access eTPU memory when the eTPUs are executing.



Configure eTPU Memory Mapping

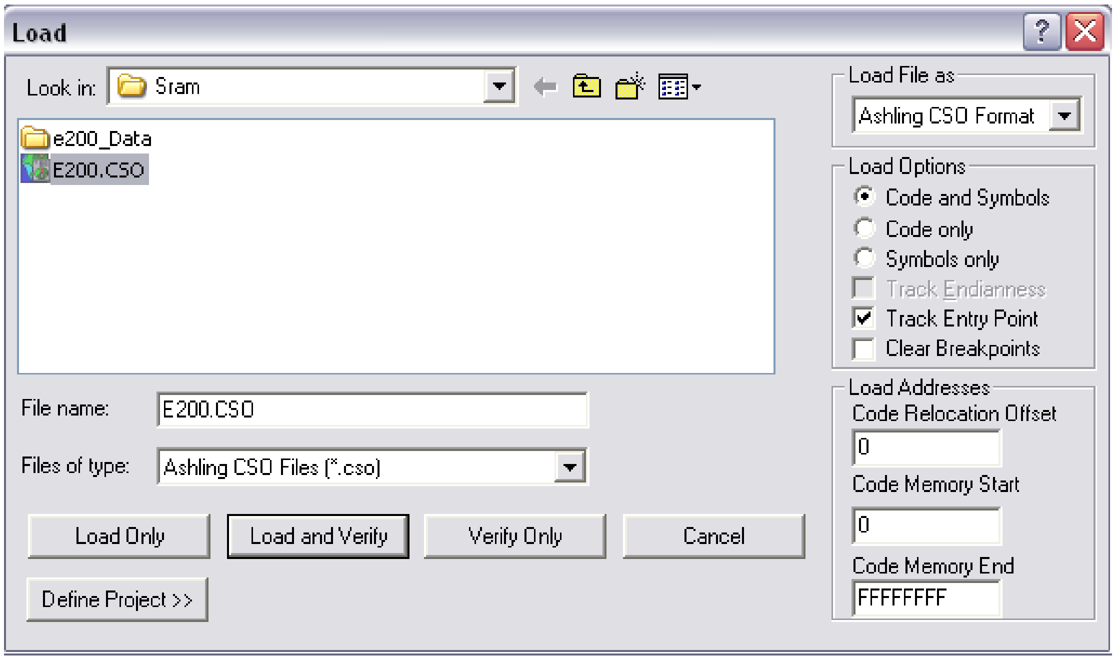
eTPU Memory Mapping

[✓] Enable	e200 Start Addr 400003A0	eTPU SCM Start Addr 0	Size 3000
[ ] Enable	e200 Start Addr 40000000	eTPU SCM Start Addr 0	Size 100
[ ] Enable	e200 Start Addr 40000000	eTPU SCM Start Addr 0	Size 100
[ ] Enable	e200 Start Addr 40000000	eTPU SCM Start Addr 0	Size 100
[ ] Enable	e200 Start Addr 40000000	eTPU SCM Start Addr 0	Size 100

Figure 9 eTPU Memory Mapping

4.2 eTPU Debugging

The example program `SRAM\e200.CSO` can be loaded using PathFinder's **File|Load** dialog as in Figure 10 below:



E200.CSO

File name: E200.CSO

Files of type: Ashling CSO Files (*.cso)

Load File as: Ashling CSO Format

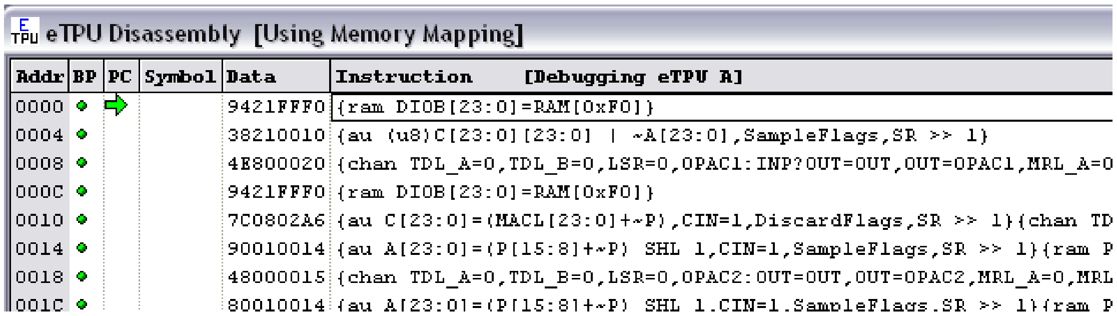
Load Options
- [✓ Code and Symbols]
- Code only
- Symbols only
- Track Endianness
- [✓ Track Entry Point]
- Clear Breakpoints

Load Addresses
- Code Relocation Offset: 0
- Code Memory Start: 0
- Code Memory End: FFFFFFFF

Figure 10 PathFinder Load Dialog

4.2.1 eTPU Disassembly Window

PathFinder provides the **eTPU Disassembly** Window as in Figure 11 below:



```
eTPU Disassembly  [Using Memory Mapping]
Addr BP PC Symbol Data     Instruction     [Debugging eTPU A]
0000 ●  ⇨         9421FFF0 {ram DIOB[23:0]=RAM[0xF0]}
0004 ●            38210010 {au (u8)C[23:0][23:0] | ~A[23:0],SampleFlags,SR >> 1}
0008 ●            4E800020 {chan TDL_A=0,TDL_B=0,LSR=0,OPAC1:INP?OUT=OUT,OUT=OPAC1,MRL_A=0
000C ●            9421FFF0 {ram DIOB[23:0]=RAM[0xF0]}
0010 ●            7C0802A6 {au C[23:0]=(MACL[23:0]+~P),CIN=1,DiscardFlags,SR >> 1}{chan TD
0014 ●            90010014 {au A[23:0]=(P[15:8]+~P) SHL 1,CIN=1,SampleFlags,SR >> 1}{ram P
0018 ●            48000015 {chan TDL_A=0,TDL_B=0,LSR=0,OPAC2:OUT=OUT,OUT=OPAC2,MRL_A=0,MRL
001C ●            80010014 {au A[23:0]=(P[15:8]+~P) SHL 1,CIN=1,SampleFlags,SR >> 1}{ram P
```

Figure 11 eTPU Disassembly Window

This Window shows disassembled code for either eTPU A or eTPU B; the eTPU may be selected using the right-mouse menu. If both eTPU engines are halted then PathFinder will read the eTPU memory directly; otherwise, it will use the e200 memory as specified in the **eTPU Memory Mapping** dialog (and this will be indicated by **Using Memory Mapping** in the title of the **eTPU Disassembly** Window).

Double-click on the ● marker in the **BP** column in the window to set a breakpoint. The marker now changes to 🔴 indicating that a software breakpoint has been set. Hardware Breakpoints (indicated by ♦ marker) can be set or cleared by pressing the Shift key and double-clicking in the BP column of the window.

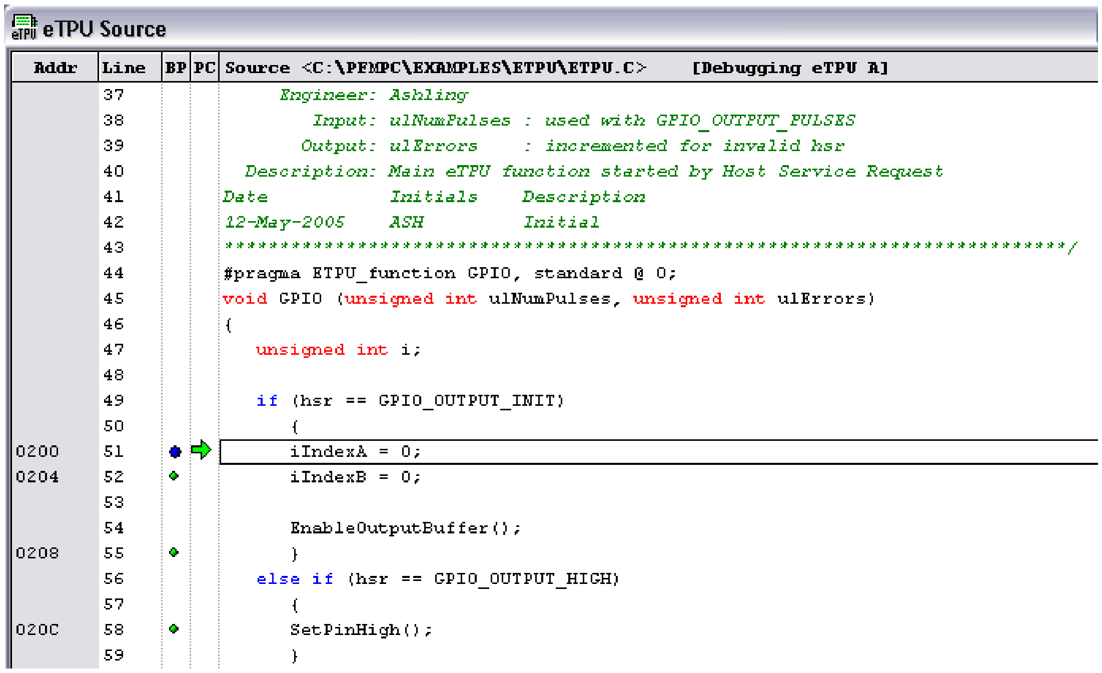
Please note the following important information about breakpoints:

- Software breakpoints 🔴 are implemented by patching eTPU program memory. They cannot be set when either of the eTPU engines is running; in addition, they will be cleared when the e200 core "copies" over eTPU code at startup (for example, when you press **Go From Reset** within PathFinder).

- Hardware breakpoints ♦ may be set when either of the eTPU engines are running and will not be cleared by a Reset. The MPC5500 supports a total of four hardware breakpoints; two for each eTPU.

4.2.2 eTPU Source Window

PathFinder provides the **eTPU Source** Window as in Figure 12 below:



eTPU Source

```
Addr   BP PC Source <C:\PFMPC\EXAMPLES\ETPU\ETPU.C>     [Debugging eTPU A]
                   Engineer: Ashling
                      Input: ulNumPulses : used with GPIO_OUTPUT_PULSES
                     Output: ulErrors    : incremented for invalid hsr
                Description: Main eTPU function started by Host Service Request
              Date           Initials    Description
              12-May-2005    ASH         Initial
              ***************************************************************************/
              #pragma ETPU_function GPIO, standard @ 0;
              void GPIO (unsigned int ulNumPulses, unsigned int ulErrors)
              {
                 unsigned int i;

                 if (hsr == GPIO_OUTPUT_INIT)
                    {
0200                iIndexA = 0;
0204                iIndexB = 0;

                    EnableOutputBuffer();
0208                }
                 else if (hsr == GPIO_OUTPUT_HIGH)
                    {
020C                SetPinHigh();
                    }
```

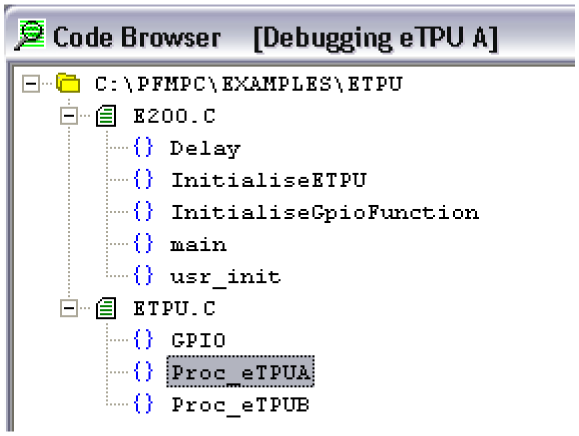
Figure 12 eTPU Source Window

This Window shows the high-level source code for either eTPU A or eTPU B; the eTPU may be selected using the right-mouse menu.

Double-click on the ♦ marker in the **BP** column in the window to set a breakpoint. The marker now changes to ● indicating that a software breakpoint has been set. Hardware Breakpoints (indicated by ♦ marker) can be set or cleared by pressing the Shift key and double-clicking in the BP column of the window.

4.2.3 Code Browser Window

PathFinder provides the **Code Browser** Window as in Figure 13 below:



Code Browser [Debugging eTPU A]

```
C:\PFMPC\EXAMPLES\ETPU
   E200.C
      {} Delay
      {} InitialiseETPU
      {} InitialiseGpioFunction
      {} main
      {} usr_init
   ETPU.C
      {} GPIO
      {} Proc_eTPUA
      {} Proc_eTPUB
```

Figure 13 Code Browser Window

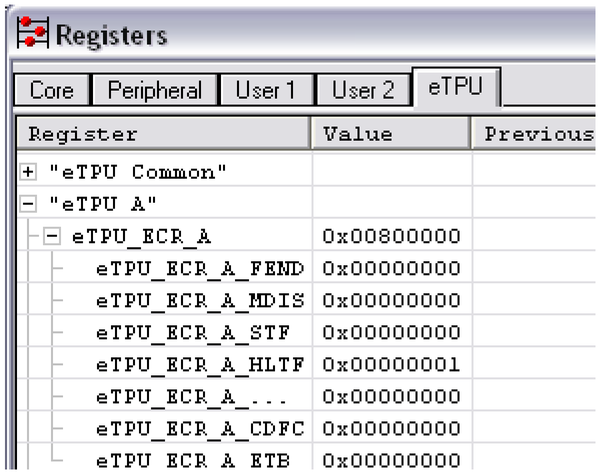
This Window shows the source files and functions within your application. For the eTPU example, both the e200 source file (e200.C) and the eTPU source file (eTPU.C) are listed in the window. Click on any function within the Code Browser to cause the appropriate Source and Disassembly windows to display that function.

Double-click on a function in the window to set a breakpoint. The marker now changes to ● indicating that a software breakpoint has been set. Hardware Breakpoints (indicated by ♦ marker) can be set or cleared by pressing the Shift key and double-clicking the function in the window.

When setting hardware breakpoints for the eTPU functions in the Code Browser window, you will need to specify whether you want the hardware breakpoint set for eTPU A or eTPU B. This is selectable from the right-mouse menu (via the **Debug engine eTPUA**, **Debug engine eTPUB** menu options). The currently selected engine is specified in the title bar of the Code Browser window.

4.2.4 eTPU Register Window

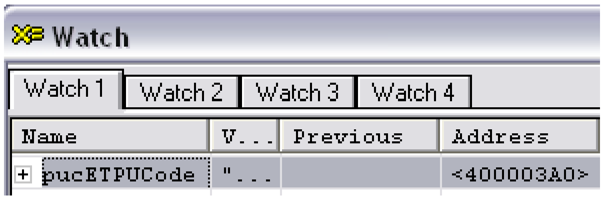
This **Register** Window shows the contents of all eTPU registers in the eTPU tab as shown in Figure 14 below:



Register	Value	Previous
+ "eTPU Common"		
− "eTPU A"		
− eTPU_ECR_A	0x00800000	
eTPU_ECR_A_FEND	0x00000000	
eTPU_ECR_A_MDIS	0x00000000	
eTPU_ECR_A_STF	0x00000000	
eTPU_ECR_A_HLTF	0x00000001	
eTPU_ECR_A_...	0x00000000	
eTPU_ECR_A_CDFC	0x00000000	
eTPU_ECR_A_ETB	0x00000000	

Figure 14 Register Window showing eTPU Registers

Our example e200 program contains the eTPU code at `0x4000-03A0` for a length of `0x3000`; you can verify this by Watching `pucETPUCode` as in Figure 15 below:

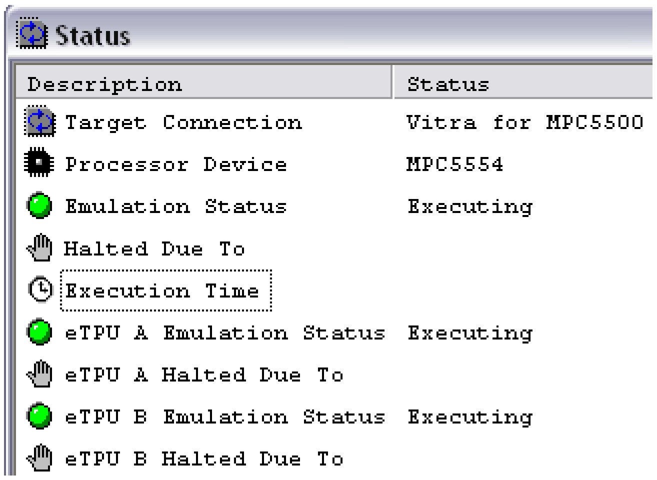


Name	V...	Previous	Address
+ pucETPUCode	"...		<400003A0>

Figure 15 pucETPUCode contains the address of eTPU code in e200 memory space

4.2.5 eTPU Status

eTPU status information is shown in the **Status** Window and the **Status Bar** as shown in Figure 16 and Figure 17 below:



Description	Status
Target Connection	Vitra for MPC5500
Processor Device	MPC5554
Emulation Status	Executing
Halted Due To	
Execution Time	
eTPU A Emulation Status	Executing
eTPU A Halted Due To	
eTPU B Emulation Status	Executing
eTPU B Halted Due To	

Figure 16 Status Window showing eTPU engine status

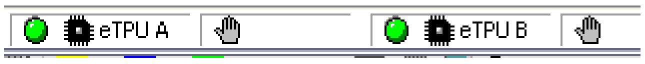


Figure 17 Status Bar showing eTPU engine status

The **Status Bar** uses the following color conventions for eTPU Emulation Status:

- Power down (grey)
- Executing (green)
- Halted in an idle state (yellow)
- Halted in an active thread (red)

4.3 eTPU run-time control

Individual eTPUs may be run, halted or stepped using the appropriate **Run|eTPU** menu. The following keyboard accelerators are also provided:

eTPU	Action	Key
A	Go/Halt	Ctrl+Shift+A
B	Go/Halt	Ctrl+Shift+B
A	Step	Ctrl+Alt+A
B	Step	Ctrl+Alt+B

Table 1.eTPU Accelerators

eTPUs may be stepped only when they are halted in an active thread. The eTPU run-time control commands can also be accessed from either the eTPU Control Bars (using the mouse) or via PathFinder's Command Window (using the command line interface) as outlined in the following section.

4.4 PathFinder's eTPU Command Window Support

PathFinder's Command Window provides full support for eTPU debugging allowing the development of scripts (or group-files) to debug/test eTPU applications. The following eTPU commands are supported:

```
  ETPUA | ETPUB  are eTPU engine specific command

ETPUx C ALL             - Clear all HW and SW BPs and WPs
ETPUx C ALL Addr        - Clear all HW and SW BPs and WPs at Addr
ETPUx C H   Addr        - Clear HW BPs at Addr
ETPUx C S   Addr        - Clear SW BPs at Addr
ETPUx C W   Addr        - Clear WPs    at Addr

ETPUx D ALL             - Display all HW and SW BPs and WPs
ETPUx D ALL Addr        - Display all HW and SW BPs and WPs at Addr
ETPUx D H   Addr        - Display HW BPs at Addr
ETPUx D S   Addr        - Display SW BPs at Addr
ETPUx D W   Addr        - Display WPs    at Addr

ETPUx S H   Addr        - Set HW BP at Addr
ETPUx S S   Addr        - Set SW BP at Addr
ETPUx S W   Addr        - Set WPs   at Addr

ETPUx G                 - Go
ETPUx HALT              - Halt
ETPUx STEP              - Step
ETPUx BREAK             - Show eTPUx Cause of break

where Addr can be: Addr, Addr T Addr, Addr L Len
```

5 For more information…

At this stage, you have completed all of the essential steps to create, compile and link an MPC5500 eTPU program, and to load, debug and execute the program using the PathFinder-MPC5500 Debugger.

You'll find full details on all PathFinder operations and commands in the appropriate Ashling User manuals. To keep your Ashling software up-to-date, check regularly for the latest software downloads at www.ashling.com/support/mpc5500 by following the link to **Download PathFinder-MPC5500.**

www.ashling.com/support/mpc5500

Ashling Microsystems Ltd
National Technology Park
Limerick
Ireland

Phone: +353 61 334466
Fax: +353 61 334477
Email: support@ashling.com

Doc: APB197-eTPUDebugging.doc