Freescale Semiconductor
Application Note

AN2821
Rev. 2, 08/2007

eTPU Host Interface

by: David Paterson
Ming Li
MCD Applications

1 Introduction

This application note discusses the enhanced Time Processing Unit (eTPU), hardware and software of the host interface, and describes the software integration.

See http://www.freescale.com/etpu for eTPU software and examples.

The eTPU is the new generation of a Time Processing Unit (TPU) by Freescale. Besides the hardware enhancement, significant improvements over TPU have been made to the accompanying software development tools; these tools make the eTPU easy to use. A high level (C) language compiler has been developed so the eTPU can be programmed using C language instead of microcode.

To program the eTPU effectively, you must have a clear understanding of how the eTPU hardware works. Coding in C, the programmer can leave the mechanics of the eTPU programming to the compiler (parameter packing, micro-instruction packing, etc.) and focus more on the

Contents

freescale semiconductor

application logic. With the help of the eTPU simulator and debugger, eTPU software can be developed much like the software for the host CPU. Productivity of software development can be significantly improved.

The introduction of the eTPU C compiler also changes the way the host interfaces to the eTPU functions. With the help of the compiler, the same symbol can be referenced by the eTPU and host software. The host software can interface with eTPU functions via application programming interface (API) functions, instead of accessing physical memory locations and registers. The host application can call these API functions to interface with the eTPU. The references to these API functions and symbols for parameters are resolved at compile time. The implementation details of the eTPU functions are hidden from the host application. This design improves the flexibility of the eTPU functions' implementation and the portability of the host application code. This application note discusses how to build the host interface for eTPU functions.

2 Overview

Host interface software adds another layer of abstraction between the host CPU and eTPU. The host interface API functions hide the complexity of the interaction between the host CPU and eTPU, providing a simple interface for host applications. Ideally, every eTPU function shall have one or more host interface API functions.

The interface software between host and eTPU facilitates three major tasks:

- eTPU hardware initialization – configure eTPU peripheral hardware
- eTPU function initialization – pass initial parameters and initiate function execution
- eTPU function run time interactive control – update function parameters and manages handshaking

After the eTPU peripheral and eTPU functions are initialized, each eTPU function can start to execute with initial function parameters. The host interface API functions need to be provided for interactive control such as parameters updating, control mode changing, etc. To update the parameter or change control mode, the host is responsible for passing updated parameters to the eTPU functions and then informing the eTPU that the function parameters have changed. If a coherent change of function parameters is required, the logic has to be built in the eTPU functions to ensure the coherency. For some eTPU functions, the interaction between host and eTPU is an essential part of the operation. In host and eTPU software, the logic is needed to manage the handshaking between host and eTPU. The interaction between the eTPU and host can be encapsulated in the host interface API functions.

The host code and eTPU code are compiled by different compilers. The host compiler is normally used to build a single code image for host and eTPU. To build eTPU code and host code together, the eTPU software building information (i.e. eTPU code image) has to be exported to the host compiler. The symbol information has to be exported from the eTPU compiler to the host compiler as well. For the Byte Craft eTPU compiler, the mechanism is implemented as a set of host interface macros. In the eTPU code, these macros are inserted to generate proper executable and symbol information for the host compiler.

3 eTPU and Host Interface Hardware

Figure 1 shows the host interface hardware for the eTPU. The host and eTPU can communicate to each other via event or by data.

The host has access to all eTPU host interface registers. When the host wants the eTPU's services, it can issue the host service request (HSR) by writing the eTPU channel control registers. After the host service request is acknowledged, a thread of eTPU code associated with this HSR is activated for execution. The eTPU code in the thread implements the functions requested by the host. When eTPU needs the host service, it can issue the interrupt request or DMA data transfer request, or generate a global exception. The events handling logic is needed in the host software to provide services to the eTPU corresponding to these requests.

The eTPU code memory (RAM) and data memory (RAM) are accessible by host and eTPU. The eTPU code memory stores the eTPU executable binary image. At the power up initialization, following the defined sequences, the host transfers the eTPU code stored in the flash memory to the eTPU code memory. During the execution, the eTPU micro-engine fetches the micro-instructions from the code memory.

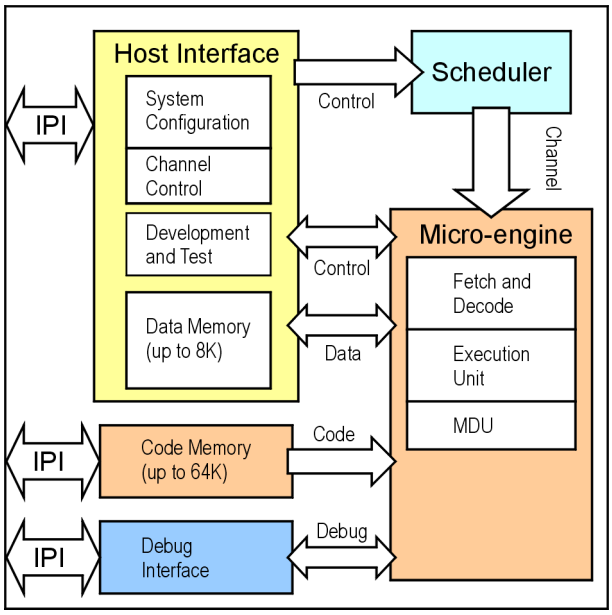

Figure 1. eTPU Host Interface Hardware

The eTPU data memory provides for data sharing between the host and eTPU engines. The eTPU can read and write the eTPU data memory in any data size (8, 16, 24, and 32 bit). However, the host can only read and write in 8, 16, and 32 bits. A virtual mirror memory space of the normal eTPU data memory, parameter sign extension (PSE) memory space, has been created to allow for converting the 24-bit data. This allows the host to read signed 24-bit data and write signed and unsigned 24 bit data.

4 Host Interface Software

The function partition between the application software and low-level driver is defined during the software architecture design. To make the software portable, the application software is normally designed to interface with the low-level driver via abstracted interface APIs. The host interface of the eTPU functions should be designed so the implementation of the low-level driver is hidden from the application software.

After the application software interface to the low-level driver is defined, the functionality of the low-level driver can be partitioned between host CPU and eTPU. The eTPU interface software running on the host CPU is an integral part of the low-level driver. It manages the details of the interaction between host and eTPU.

The eTPU code and host interface code are compiled by two different compilers. To resolve the eTPU code symbol reference in the host code, it is necessary to export the symbolic information from the eTPU compiler to the host compiler. Similarly, to build a single executable image, it is necessary to export eTPU code image to the host compiler. Several types of information are exported from eTPU_C through host interface files, generated from #pragma write statements during compilation. For detailed information on #pragma write, see the eTPU_C documentation.

4.1 Initialization Overview

The eTPU initialization is an important part of the host interface. At the power up, the host has to configure the eTPU peripheral properly before the eTPU function can be executed. The eTPU initialization process is accomplished by host CPU and eTPU. During the initialization, the functional partition between host CPU and eTPU is as follows:

Host responsibility:

- eTPU module initialization
- eTPU channel configuration
- Providing initial eTPU function parameters
- Initiating the eTPU function execution

eTPU Responsibility:

- Responding to the HSR
- Transitioning into the initialization state

The eTPU initialization can be broken down into three steps: eTPU module initialization, eTPU channel initialization and eTPU function initialization. The following sections discuss the interface design and the data exchange between host CPU and eTPU for each step of initialization.

4.2 eTPU Module Initialization

At the power up initialization, the eTPU peripheral hardware is configured by the host. The eTPU module initialization includes the following steps:

1. Initialize eTPU global registers
   - eTPU MISC compare register (ETPU_MISCCMPR)
   - eTPU module configuration register (ETPU_MCR)
   - eTPU time base configuration register (ETPU_TBCR)
   - eTPU STAC bus configuration register (ETPU_STACR)
   - eTPU engine configuration register (ETPU_ECR)
2. Load eTPU code to eTPU code RAM
3. Copy initial values of eTPU code global variables to eTPU data memory

Most of the information required to configure the eTPU global registers are not dependent on the eTPU software implementation (time base, clock frequency, entry table address, etc.). The configuration information is determined during the host peripheral configuration and resource allocation. Only the MISC value depends on the actual eTPU software implementation; it has to be exported to the host program after the eTPU code is compiled. The eTPU_C compiler provides a macro (::ETPUmisc) to calculate the MISC value of the eTPU code image.

The eTPU code image is generated when the eTPU C program is compiled. To use this code in the host program, the eTPU code image can be exported as an array of constant values. The eTPU_C compiler provides a macro (::ETPUcode) to generate and export the eTPU code image. The eTPU code image constant array is suitable to be included in the host source code. The host compiler locates the eTPU code image array at host source code compile time. Because the eTPU micro-engine can only fetch micro-instructions out of eTPU code memory (RAM), at the power up initialization, the eTPU code image has to be loaded to eTPU code memory.

As it is in the standard C syntax, when the eTPU code is compiled, the global variables are allocated to the eTPU data memory and the initial values are assigned to the corresponding memory locations. The standard C syntax requires that all global variables are declared outside of any function. For eTPU, this means that all global variables have to be declared outside of any execution thread. Because only the code in a thread can be executed on eTPU, the global variable initial value assignment statements are not executed. The global variables cannot be initialized in the eTPU code; they must be initialized by the host.

The software statements have to be added to the host code to initialize the eTPU global variables. The initialization values have to be exported from the eTPU compiler to host compiler. The eTPU_C compiler provides a macro (::ETPUglobalimage) to capture initial values for all global variables and exports them as a constant array. At power up initialization, the global variable initial values are loaded to the eTPU data memory for global variables.

The host interface macros have to be added to the eTPU code in #pragma write statements to export the eTPU software information to the host compiler. These statements generate header files that contain the MISC value, eTPU code image and global variable memory image.

4.3 eTPU Channel Initialization

After the eTPU module is configured, each channel on the eTPU module can be configured. The eTPU channel initialization includes the following tasks:

- eTPU channel configuration registers initialization
  - Assign eTPU function to a channel
  - Configure interrupt/DMA/Output enable
  - Select eTPU function entry table encoding
  - Assign the function frame to a channel
  - Set-up channel priority
- eTPU channel status control register initialization
  - Set-up eTPU function mode

The eTPU channel assignment and the channel priority determination are a part of the host software architecture design. They are independent of the eTPU functions implementation. The information for the configuration is provided by the host software design.

During the channel initialization, a section of eTPU data memory is assigned to each channel. This memory section is the function frame. The function frame contains all the function parameters and static local variables used by the eTPU function. The starting address of the function frame is assigned to the channel at initialization. The function frame assignment can be static or dynamic. Dynamic allocation assigns the function frame to the channel based on the next available memory space. The availability of the eTPU data memory depends on the number of functions that have been assigned and the number of parameters the function is using. Dynamic allocation can reduce the eTPU data memory consumption by minimizing unused memory holes. To allocate the function frame dynamically, the host must know the function frame consumption by a particular eTPU function. The eTPU_C compiler provides a macro (::ETPUram) to report the number of parameters and static local variables used by a function at compile time.

After the host interface macros are added to the eTPU code, the #pragma write statements generate a header file that contains all the eTPU function configuration and software symbol information at compile time. The header file can be included in the host interface code to resolve symbol references.

4.4 eTPU Function Initialization

The eTPU function initialization is the last step of the eTPU initialization process. During the eTPU function initialization, the host is responsible for passing the parameters to the eTPU functions and initiating the eTPU function execution by issuing a host service request. After the host service request for initialization is recognized, the eTPU transitions to the initialization state.

Unlike in the host CPU, the eTPU function parameters passed from host are not placed on the stack. Instead, memory in the function frame is allocated to accommodate every function parameter. The host passes the eTPU function parameters by writing directly to the eTPU function frame. The host needs to know the function frame address for each channel, as well as the data type and address offset for every parameter. The function frame address can be derived by reading the eTPU channel base address register.

The eTPU_C compiler provides the host interface macros to export the offset of each function parameter; use them in #pragma write directives to export this information.

The function parameters can be 8-bit, 16-bit, or 32-bit. The eTPU compiler can allocate function parameters at 8-, 16-, 24-, or 32-bit boundaries. To pass 8-bit or 16-bit parameters, the host can directly write to eTPU data memory.

Most eTPU data registers and timers are 24-bit. To pass 24-bit eTPU function parameters, the host needs to pass a 32-bit parameter to the eTPU. Because the host cannot access eTPU data memory on the 24-bit boundary, the host code needs to realign the parameter to the 32-bit address boundary before writing it to the function frame. It is the responsibility of the host to ensure the function parameters are within proper range. It is also the responsibility of the host when writing the 24-bit parameter to ensure that the upper byte on the function frame is not corrupted. Similarly, when reading a 24-bit return value from the function frame, the host code must mask the upper byte before returning the correct 24-bit value. To simplify the interface code, it is recommended to access the 24-bit function parameter by using PSE memory space.

4.5 eTPU and Host Interactive Control

After the eTPU function is initialized, it starts execution based on the initial parameters and input/output conditions. The API for the host application code updates the function parameter or changes the control mode. Similarly, host software must provide proper logic to manage the eTPU interrupt or DMA requests.

Some eTPU functions require host or DMA service. The eTPU software can write the CIRC bits in the channel interrupt and data transfer request register to send the request to the host or DMA. The interrupt service routine must be added and the DMA channel must be configured in the host code to respond to the eTPU request.

The host interface software has to provide functions to update eTPU function parameters or change the control mode during the normal operation. Similar to the function initialization API, the interface API function needs to check the validity of the parameters, write them to the eTPU data memory, and then issue the host service request to inform eTPU that the parameters are newly updated.

5 Software Integration

The eTPU code and host CPU code are compiled and linked separately. The eTPU code needs to be built first to generate and export the eTPU code image and parameter symbol information. The host code needs to include these files properly to resolve all the symbol reference between eTPU and host code. This software build dependency can be added easily to the makefile to ensure the proper sequence.

6 Conclusion

The benefit of the host interface design is to isolate any hardware dependency from the application software by means of the host interface API functions. In the eTPU host interface design, all the interactions between host and eTPU are encapsulated in the interface API functions. With this interface design, the implementation of the low-level driver can be hidden from the host application.

Numerous examples are available in the general set and APIs available from Freescale.com.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2821
Rev. 2
08/2007

freescale™
semiconductor