

QL5064 Hardware Abstraction Layer

Early Access Release 8,
08 February 2007

Introduction

This document describes the hardware abstraction layer for use with the Sundance SMT417, SMT498 and other PMC cards produced by Sundance DSP. It includes descriptions of the functions available in the package as well as details of the procedures to be used when setting up and working with these boards.

Overview

The SMT417 is a Dual DSP PMC/XMC module. Typical use of these modules is in VME64 and cPCI carrier boards. Several bus adapter boards are available which provide a PMC site to a PCI-X slot on ATX form-factor platforms. There are also adapters that provide an XMC site to PCI-e slots on ATX form-factor platforms as well.

The interface is implemented as a user-space adaptation layer. Access to the PCI configuration space, and other system related interfaces is implemented by either WinDriver or the open-source pciutils packages. The QL5064 Hardware Abstraction Layer uses these APIs to configure the bridge to map board memory regions into user address space. All interactions between the Host CPU and the SMT417 then become memory-mapped I/O.

The QL5064 Hardware Abstraction Layer exposes these memory mapped regions to the application by way of a data structure. Additionally, several subroutines are provided that automate common tasks such as enumerating the number and location of SMT417 boards, performing board reset, and simple, blocking, link port I/O.

Requirements

The following requirements must be met for a system to be able to use the QL5064 Hardware Abstraction Layer Package:

- SMT417, SMT498 or other Sundance module based on the QL5064 bridge.
- TwinIndustries Xtend1000 or Technobox 4366 PMC/PCI-X adapters¹.
- Microsoft Visual C++ 6.0.
- Windows XP SP2, 2003 Server, or other Win32 operating system.

¹ Other adapters may be supported through testing and integration, contact Sundance for details.

Packaging

The software is provided as the following files:

- `ql5064_hal.h` contains the API definitions as described in this *User Guide*, and must be `#included` into user applications.
- `ql5064_pci.h` contains definitions of the registers, bit masks and meanings of the memory-mapped registers available. The application may make use of these registers if it chooses to control the modules directly.
- `smt417.h` contains definitions for the SMT417-specific resources used.
- `smt_types.h` contains definitions for C99 types for those compilers which are non-compliant. It is included into user applications by `smt_io_devinfo.h`; you do not need to include it yourself.
- `smt_io_devinfo.h` contains definitions for the resource management structure which holds the enumeration information of SMT417 boards that have been attached.
- `smt_sema.h` contains definitions for a multi-platform semaphore/mutex API that is based on the same API in 3L/Diamond. It is included into user applications by `smt_io_devinfo.h`; you do not need to include it yourself.
- `ql5064_hal.lib` is the library containing the subroutines implementing the abstraction layer, and must be linked into user applications.
- `ql5064_exec.exe` is a Win32 HOST application which calls on the HAL to perform board enumeration, reset and program loading.
- `ql5064_cp_test.exe` is a HOST application which calls on the HAL to perform I/O operations between the HOST and the processors on the SMT417 module. It can reset/load programs, the FPGA, etc. It expects that whatever program/bitstream is loaded or resident on the SMT417 will be capable of being a data source/sink/loopback.
- `BlinkGPIO.hex` is an application you can load on either of the DSPs of the SMT417 module. When run, it continuously cycles the 2 LEDs available to each the DSP on the module. It is a basic system load application.
- `BootAndBlinkGPIO.hex` is an application you can load to DSPA of the SMT417 module. When run, it first loads 'BlinkGPIO.hex' to DSPB via a comport link, then executes the LED link test. It shows how to load DSPB under the control of DSPA as well as several of the comport API routines.
- `CPLoopback.hex` is an application you can load on either of the DSPs of the SMT417 module. When run, it loops back all data received on any of its link ports. It also toggles the 2 LEDs for every word processed... It is a good test to use for bi-directional data integrity.

Other programs may be provided and are described in the `readme.txt` files that are dispersed throughout the software distribution.

Target Software Preparation

In order to be loaded onto a Sundance (DSP or FPGA) module, an application must be prepared for loading. The simplest method is to use 3L/Diamond to create an .app through a process of *configuration*. Once an .app is obtained, it can be presented to t328_exec.exe for loading onto the system.

[NOTE: the method above is not available in this release of the BSP]

Another method is to use CCS to generate an executable application image called an .out file. This file can be loaded via TI-DSP JTAG interfaced through the JTAG adapter board. This provides convenient access to a Std-C system interface <stdio.h> through TI's CCS drivers.

In order to properly load the program, your applications should respect a reserved region of internal memory which is assigned for the interrupt vector , bootloader and hex loader programs:

```
MEMORY
{
    VECT      (RWX):  o = 00000000h  l = 00000400h
    RESV1    (RWX):  o = 00000400h  l = 0000a000h
    BOOT     (RWXI): o = 0000a400h  l = 000000a0h
    PMEM     (RWXI): o = 0000a4a0h  l = 000f5b50h
    EMIFB    (RW):   o = 60000000h  l = 10000000h
    SDRAM    (RWX):  o = 80000000h  l = 40000000h
}

SECTIONS
{
    boot: {
        *(.text:_c_int00)
    } > BOOT
    .text: {
        *(.text)
    } > PMEM
    .const > PMEM
    .data > PMEM
    .bss > PMEM
    .cinit > PMEM
    .pinit > PMEM
    .stack > PMEM
    .cio > PMEM
    .systemem > PMEM
    .switch > PMEM
}
```

Once loaded, your application may make use of these regions, but it should not place initialized sections in them. The hexloader for CCS applications uses 0xA400 as the entry point address. The linker commands above describe a method of contriving the C initialization routine to be placed at address 0xA400 for the bootloader.

CCS applications developed above can be loaded via the t328_exec.exe application. In order to do this, the .out files are first converted to .hex via the TI utility hex6x, in the Intel-86 Hex format. This file is loaded via link port I/O after a board reset.

To convert a .out to a .hex please use the following command:

```
hex6x -o file.hex -romwidth 32 -i file.out
```

This creates an ASCII memory file which can be loaded by resetting the system and sending the contents of the file verbatim down any link port connected to a processor. The reset operation invokes the FLASH bootloader on the SMT417 which configures the system registers and loads a hexloader stub to accept the uploaded program.

Examples for developing such applications are provided in the `examples/ccs/` folder of the BSP. Please review these.

You may also choose to load your application into FLASH and have it be loaded automatically upon board reboot. To do this, you need to also create a .hex file (as described above), but with the following restrictions:

1. You may not use the `<stdio.h>` interfaces or any other interfaces which assume a HOST connection.
2. You must add an initialization table to address 0x0 so that the bootloader can properly handle your application.
3. If your application is intended to run on DSPB, it must also contain the code to configure the EMIF on DSPB as part of this bootstrap.

An example of developing such applications is provided in the source to the `load417.hex` application located on the BSP at `src/smt417/hexloader/`.

Once you do this, you have a .hex file which can be loaded into the FLASH for bootloading via the `smt417FlashTool.out` under CCS/JTAG.

Application Programming Interface (API)

API Overview

User code accesses the QL5064 Hardware Abstraction Layer through definitions contained in the header file `ql5064_hal.h`, as follows:

```
#include "ql5064_hal.h"
```

The program making use of the QL5064 Hardware Abstraction Layer must make exactly one call to the `ql5064_init()` function for **each** board before using any other API functions. Callers provide the address of a `ql5064_priv_t` structure to receive the memory mapped registers and resource management keys for that board instance. The `ql5064_init()` requires the desired SMT417 module instance index be provided. This allows multiple SMT417 modules to be handled under the same API.

Once the QL5064 Hardware Abstraction Layer has been initialized, applications will normally use the link port I/O routines to read and write data to the processor(s) on the SMT417.

When all application operations have been completed, the application must call `ql5064_fini()` to signify that they no longer require use of the QL5064 Hardware Abstraction Layer for that particular module instance.

Enumeration

Prior to calling `ql5064_init()`, the application may desire to determine what boards are available. It can do this via the `ql5064_enumerate()` function. This function will populate an array of `smt_io_devinfo_t` structures which among other things indicates the instance and geographic location of each SMT417 module detected.

Examples of the use of the `ql5064_enumerate()` and `smt_io_devinfo_t` structure, can be found in the supplied `ql5064_cp_test.c` unit test source file.

API Details

Each function provided in the API is now described on a separate page. The function's synopsis is followed by a description of its function, along with common conditions under which it may fail.

For description of the data structures, please refer to the `ql5064_hal.h` file itself.

Function ql5064_init

Synopsis

```
bool ql5064_init(ql5064_priv_t *priv, uint32_t instance );
```

Definition

The user task making use of the QL5064 Hardware Abstraction Layer must call this function exactly once before calling any other API functions for any given module.

The user task is responsible for maintaining memory space for the `ql5064_priv_t` structure for the duration of the session (until `ql5064_fini()` is called).

Following a successful return, the other functions in the API that use a `ql5064_priv_t` pointer may be called:

- `ql5064_reset()`
- `ql5064_link_in()`, `ql5064_link_out()`
- the three address space mappings are valid for
 - bridge configuration registers (`priv->config.map.bar0`)
 - CPLD and FPGA link ports (`priv->dpr.map.bar1`)
 - FPGA general purpose regions (`priv->io.map.bar2` to `bar4`)

§ NOTE: these regions are firmware specific

Common Error Conditions

`false`

Status information is output onto `stderr`.

Function ql5064_fini

Synopsis

```
void ql5064_fini( ql5064_priv_t *priv );
```

Definition

The user task making use of the QL5064 Hardware Abstraction Layer must call this function when it no longer requires use of the file system facility. This may be used to tidy resources associated with the task. It is an error to call any API functions using the same `ql5064_priv_t` [except another `ql5064_init()`] after a call to `ql5064_fini()`.

Common Error Conditions

None.

Function ql5064_reset

Synopsis

```
void ql5064_reset (ql5064_priv_t *priv );
```

Definition

Resets the SMT417 module.

This operation will typically result in the module bootloading, configuring its built-in FPGA design and preparing to accept a DSP program on its link ports.

Common Error Conditions

None.

Function ql5064_link_in

Synopsis

```
size_t ql5064_link_in( ql5064_priv_t *priv, size_t len, void *buf_p, int port );
```

Definition

Reads `len` bytes *from* the link port specified by `port` to `buf_p`. The function blocks until complete. `len` must be an integral number of 4 bytes.

Returns the number of bytes actually read.

Common Error Conditions

None. The function may not return if the application on the target board does provide enough data. A mechanism to abort the transfer is provided by `ql5064_link_reset()`. Therefore, the number of bytes returned may not always equal the number requested.

Function ql5064_link_out

Synopsis

```
size_t ql5064_link_out( ql5064_priv_t *priv, size_t len, void *buf_p, int port );
```

Definition

Writes `len` bytes *from* `buf_p` to the link port specified by `port`. The function blocks until complete. `len` must be an integral number of 4 bytes.

The return value is the number of bytes actually written.

Common Error Conditions

None. The function may not return if the application on the target board does provide enough data. A mechanism to abort the transfer is provided by `ql5064_link_reset()`. Therefore, the number of bytes returned may not always equal the number requested.

Function ql5064_link_reset

Synopsis

```
void ql5064_link_reset( ql5064_priv_t *priv, int mask, int port );
```

Definition

Resets the link port specified by `port`. The function can affect either the outbound or the inbound side of the port. `mask` determines which direction as follows:

- 0x1 indicates the outbound side (PCI to FPGA/DSP)
- 0x2 indicates the inbound side (FPGA/DSP to PCI)

Common Error Conditions

None.

Function ql5064_fw_load

Synopsis

```
bool ql5064_fw_load( ql5064_priv_t *priv, FILE *fp );
```

Definition

Causes the FPGA on the SMT417 module to be reprogrammed with the bitstream specified by the open file pointer.

Common Error Conditions

`false`: The FPGA was not reprogrammed. Additional information is given by `errno`.

Function ql5064_set_trace

Synopsis

```
void ql5064_set_trace( FILE *fp, const char *path );
```

Definition

Activates debugging facilities. Trace output can be specified either as an open file pointer (stdout, stderr, etc. or a file), or as a path to an output log file. In the case of the log file, the file will be opened when ql5064_init() is called on the first instance and closed when ql5064_fini() is called for the last instance.

Note, driver build options may affect the amount and nature of the output produced.

Common Error Conditions

None.