基于AM335X平台的Ethercat实现

Denny Yang
TI嵌入式处理器技术支持工程师

摘要

EtherCAT 是开放的实时以太网通讯协议，最初由德国倍福自动化有限公司研发。EtherCAT 具有高性能、低成本、容易使用等特点，目前在工业领域有着广泛的应用。TI 的 AM335X 是一款基于 ARM CORTEX A8 核心的应用处理器。内部集成基于可编程实时单元的工业通信子系统（PRU-ICSS），此系统独立于 ARM 处理器工作。PRU-ICSS 支持 EtherCAT 协议，TI 免费提供基于 PRU-ICSS 的 EtherCAT 从站例程。目前 TI 不提供 EtherCAT 主站例程，用户可以自行开发主站或者寻找第三方支持，也可使用开源的主站协议栈。IGH EtherCAT 主站是一套开源的 EtherCAT 协议栈，有较大参考价值。本文将介绍 EtherCAT 从站范例和 IGH EtherCAT 主站到 AM335X 平台的移植与测试。

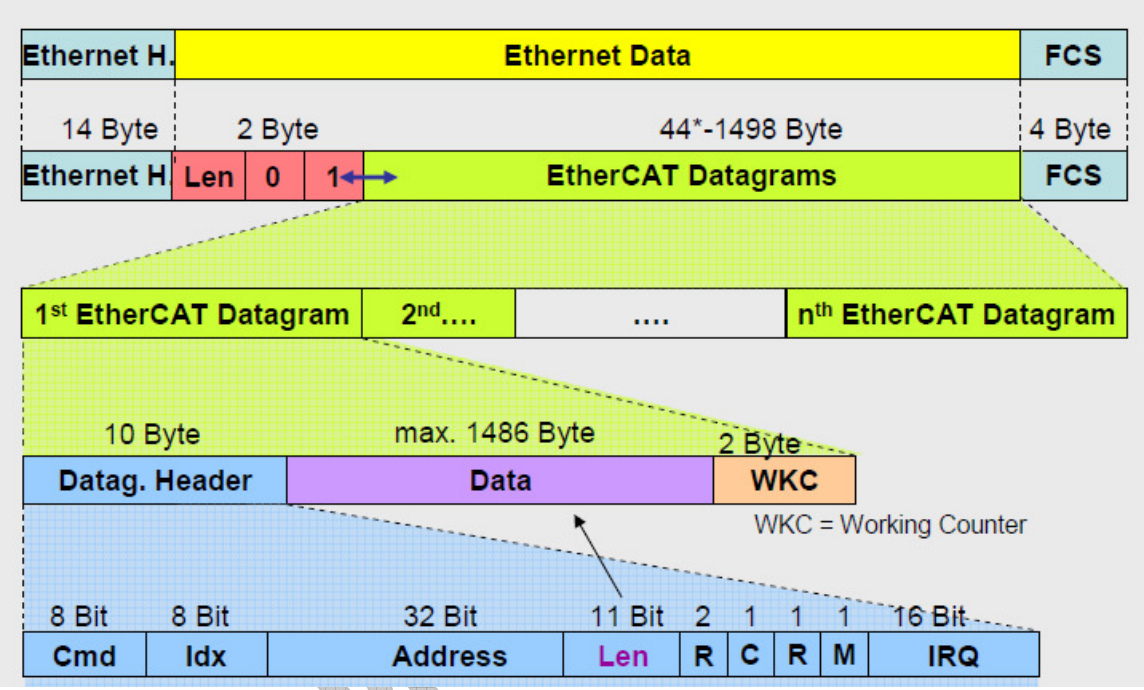
目录

图

1 EtherCAT 简介

EtherCAT（以太网控制自动化技术）是一种用于确定性以太网的高性能工业通信协议，它扩展了 IEEE 802.3 以太网标准，使得数据传输中具有可预测性定时及高精度同步等特点。这个开放性标准作为 IEC 61158 的组成部分，常用于机械设计及运动控制等应用中。EtherCAT 采用标准的 IEEE802-3 以太网帧，帧结构如图 1。EtherCAT 协议直接用标准以太网的帧格式传输数据，并不修改其基本结构。

图 1 EtherCAT 数据帧

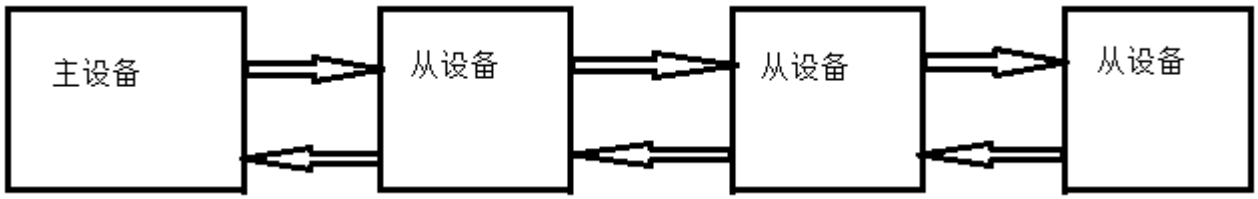


EtherCAT 实现了 CANOpen 协议，在 CANOpen 中周期性的数据通过 PDO（过程数据对象）来传输，PDO 优先级比较高用于实时传输。非周期性的数据比如配置参数等则通过 SDO（服务数据对象）来传输。

每个 PDO 都包含单个或多个从设备的地址，这种数据加地址的结构（附带用于校验的传输计数位）组成了 EtherCAT 的报文。每个 Ethernet 帧可能包含数个报文，而一个周期中可能需要多帧来传送所需的所有报文。

传统的以太网通信解决方案从站先接受以太网数据包，然后解释和复制过程数据，最后转发数据。而 EtherCAT 以太网帧在特殊的硬件模块的帮助下可以实现在传输的同时被处理。在每个从节点都有 FMMU（现场总线存储管理单元），FMMU 会对经过的数据包进行地址分析，发现是本节点的数据就会读取，同时报文转发给下一个设备。同样在报文通过的时候也可以插入数据。读取/插入/转发数据的整个过程报文只有几纳秒的延迟。如图 2 所示，设想以太网的帧就像行驶中的火车，EtherCAT 报文是每节火车车厢，PDO 数据的比特就是车厢内的乘客，这些数据可以被提取并插入到合适的从设备中。整辆火车不停止地穿越所有从设备，在末端从设备处又掉头，重新反向穿越所有从设备。

图 2 EtherCAT 数据传输



2 EtherCAT 主从站软硬件简介

2.1 TI EtherCAT 从站介绍

2.1.1 从站硬件介绍

TI 的 EtherCAT 从站范例是运行在基于 AM3359 的 ICE（Industrial Communications Engine）板上的。这款开发板是专门针对工业通信领域而设计的，使用的 AM3359 主芯片拥有 ARM Cortex-A8 的处理器和 PRU(可编程实时单元)。此开发板通过 PRU 可以支持以下工业通信协议，PROFIBUS、CANOpen、Ethernet/IP、EtherCAT、PROFINET 等。详细介绍可以参考以下链接：
http://www.ti.com/tool/tmdxice3359

2.1.2 从站软件介绍

AM335X PRU 的 EtherCAT 从站的实现，可以通过下载 am335x_sysbios_ind_sdk_win32setup_xx（http://www.ti.com/tool/sysbiossdk-ind-sitara）软件包获取其代码。TI 的 EtherCAT 从站基于 PRU-ICSS 实现，具有速度快可扩展性强等特点，并且通过了 EtherCAT Conformance Test 认证。

下载安装 am335x_sysbios_ind_sdk_win32setup_xx 后会得到一个目录 am335x_sysbios_ind_sdk_xx，此软件包包含两个 EtherCAT 例程：Simple Demo Application 和 Full Feature Demo Application。Simple Demo Application 是基于静态预编译的 EtherCAT 协议栈库(ecat_slave_stack.lib)。它允许对 SSC5.01 协议栈进行有限的开发，支持 32 位的输入和输出。Full Feature Demo Application 使用完整的 SSC5.01 协议栈，TI 不能发布 SSC5.01 的代码，用户需要自行到 ETG 官方网站上下载 SSC5.01 代码，并把代码通过补丁的方式加入到 Full Feature Demo Application 工程。这两个 DEMO 都可以接收主站发来的数字量，使用这个数字量来控制 ICE 板上的 LED 灯，还可以采集板上的开关量，并把开关量发回给主站。

从站可以通过 CCS5 导入编译，am335x_sysbios_ind_sdk_1.0.0.8\sdk\examples\ethercat_slave 这个目录是 Simple Demo Application，am335x_sysbios_ind_sdk_1.0.0.8\sdk\protocols\ethercat_slave\ecat_appl 这个目录是 Full Feature Demo Application。这两个工程都有四组编译配置：

1） Debug，测试版本。此版本支持实时分析和日志。

2） IRAM，这个版本编译出来的是 Thumb 指令，此版本完全在片内存储器中运行，无需 DDR 支持。运行此版本需要把 PRU 固件烧到 SPI Flash 中，可以通过 CCS 使用

am335x_sysbios_ind_sdk_1.0.0.8\sdk\tools\flashing tools\SPI_Flash\pre_built\isdk_spi_flasher.out 这个工具把 sdk\protocols\ethercat_slave\firmware 目录下的 ecat_frame_handler_flash.bin 烧到 SPI 地址 0x98000，ecat_host_interface_flash.bin 烧到 SPI 地址 0x9C000 。

3） NOR，此版本编译出来的程序会在 NOR Flash 和片内存储器中运行，无需 DDR 支持。

4） Release，此版本无实时分析和日志支持，在 DDR 中运行。

编译完成后把生成的.out 文件通过 JTAG 工具下载到 ICE 板上执行，系统开始运行会点亮 LED2，4，6，7。通过网线直连 ICE 和 PC，并在 PC 上安装 TwinCAT 软件可以对 EtherCAT 从站进行测试。详细测试步骤可以参考：
http://processors.wiki.ti.com/index.php/Configuring_TwinCAT_For_AM335x 。

如需要编译 Full Feature Demo Application 则需要下载 SSC5.01 源代码并复制到 am335x_sysbios_ind_sdk_1.0.0.8\sdk\protocols\ethercat_slave\ecat_appl 这个工程中，然后进行编译。详细步骤可以参考 am335x_sysbios_ind_sdk_1.0.0.8\sdk\docs\ AM335x SYSBIOS Industrial SDK 01.00.00.08 User Guide 第 13 页。

2.2 EtherCAT 主站软硬件介绍

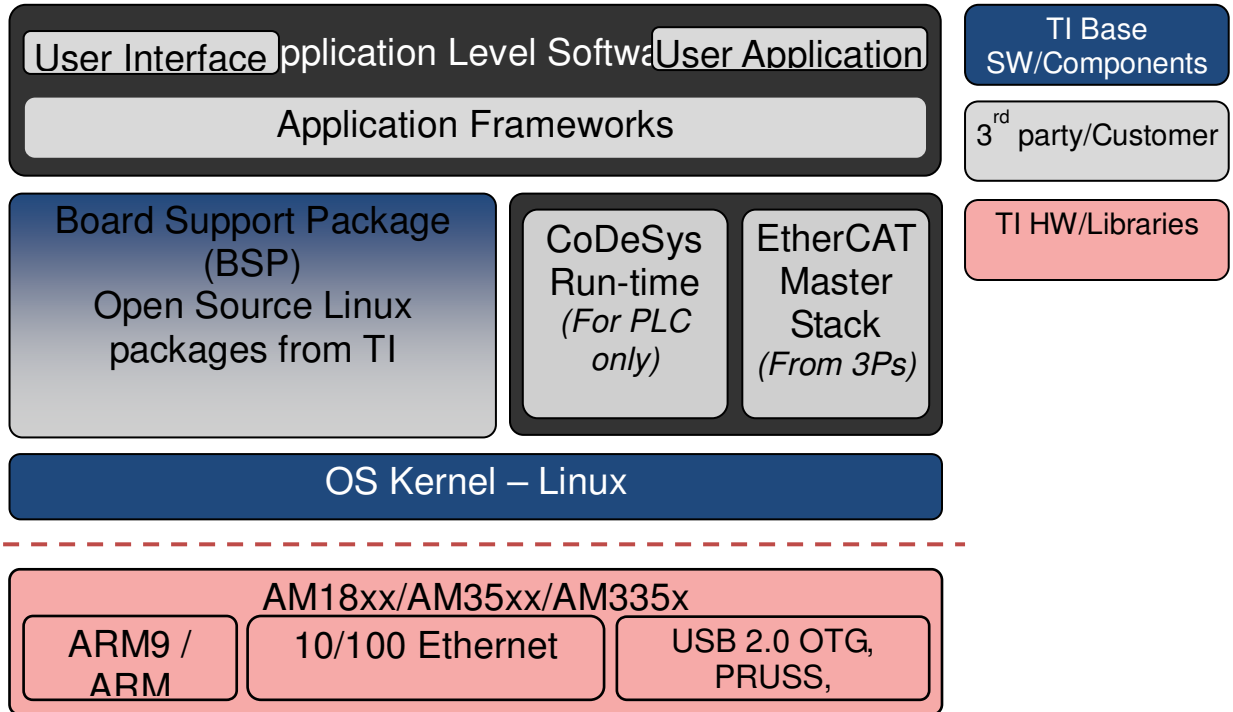
2.2.1 主站硬件介绍

本文使用开源的主站软件，理论上只要是跑 LINUX 的平台并且拥有标准一个网口都可以运行，本文选择了基于 AM335X 的 BeagleBone 作为开源软件的目标平台。BeagleBone 是 TI 提供的一套低成本的 AM335X 开发参考设计，具体介绍可以参考一下链接：http://www.ti.com/tool/beaglebn

2.2.2 主站软件介绍

目前 TI 提供基于 AM335X 的板级软件包支持，暂不提供 EtherCAT 主站的协议栈等支持。如下图所示，主站部分软件需要客户自己开发，或者使用第三方的软件协议栈（Koenig, Acontis 等），还可以使用开源的协议栈。本文第三章是采用开源的 IGH EtherCAT 协议栈来实现主站的。

图 3 TI Sitara SDK 的软件构架



3 IGH EtherCAT 主站简介和移植

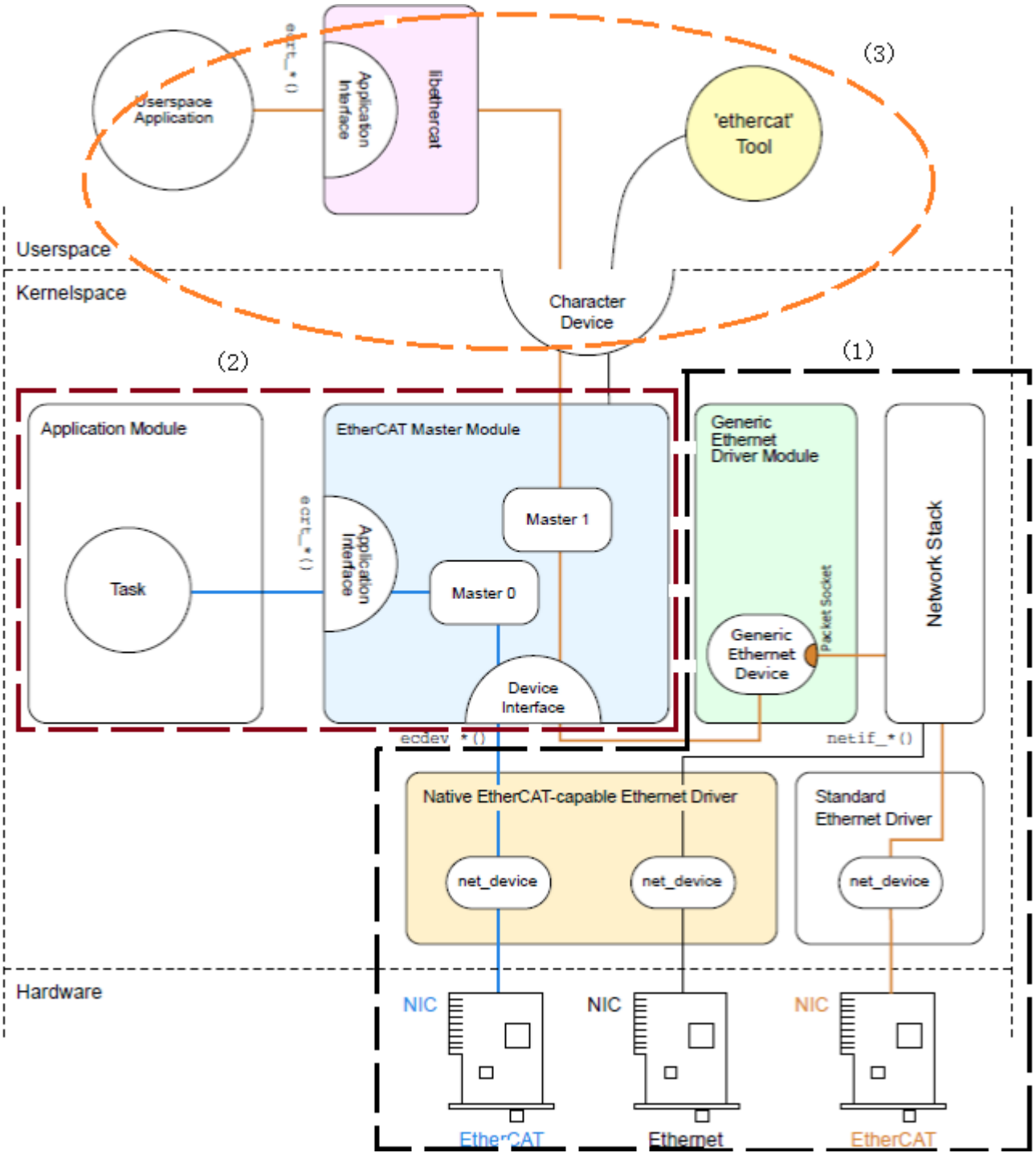
IGH EtherCAT Master 是一套基于 Linux 的开源 EtherCAT 主机软件，包括网卡驱动、内核层的主站模块和应用程序、用户层工具和支持库。图 4 的三个虚线部分示意了 1）、2）和 3）。

1） 网络设备部分。EtherCAT 软主站不使用特殊硬件，而是普通的网络设备(如 PCI 网卡)。内核层中含有网络设备模块，但为了实现 EtherCAT 协议，需要对普通网络设备驱动进行修改。IGH 在此软件版本 1.5 之后提供一个通用的网卡驱动，可以不用修改现有驱动就能支持 EtherCAT 的主站。

2） 主站模块与应用程序。为了保证实时性的要求，EtherCAT 主站模块与应用程序模块均在内核层。其中 EtherCAT 主站模块实现全部的协议解析、任务调度并为网络设备与应用程序提供函数接口。应用程序模块最终实现对各种自动化系统的控制，由用户根据具体的控制对象和控制要求进行编写。用户可以通过编写内核模块的方式实现应用程序，也可以在用户空间通过库对设备节点进行操作来实现应用程序。

3） EtherCAT 工具。该程序提供了各种可以在 Linux 用户层运行的命令。可以直接实现对从站的访问和设置。如：设置从站地址、显示总线配置、显示 PDO 数据、读写 SDO 参数等。由于用户层无法直接访问内核层的数据，因此需要构造 Linux 字符设备。通过对字符设备的访问间接实现与 EtherCAT 主站模块的通汛。

图 4 IGH EtherCAT Master 结构



从http://www.etherlab.org/en/ethercat/网站上下载最新的版本，本文作者下载 ethercat-1.5.1.tar.bz2，放到 Ubuntu 虚拟机环境中，解压缩软件包可以获得 ethercat-1.5.1 目录。建议同时下载文档 ethercat-1.5-6129a5f715fb.pdf，这个文档有对此软件包的详细描述。首先要安装 AM335X 的编译环境，可以参考 TI 的相关文档。主站编译步骤如下：

1) 使用控制台进入主站目录后，用如下命令配置编译系统：

```
./configure --prefix=/home/denny/ethercat/ethercat-1.5.1/output  --with-linux-
dir=/home/denny/ti-sdk-am335x-evm-05.07.00.00/board-support/linux-3.2.0-psp04.06.00.10
--enable-8139too=no  --enable-generic=yes  CC=arm-none-linux-gnueabi-gcc --host=arm-
none-linux-gnueabi
```

2) make

3) make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- modules

4) mkdir output/modules

5) make doc

此命令会在 ethercat-1.5.1\doxygen-output 目录下生成库函数的说明文档。

6) make install

make install 会在当前目录生成 output 文件夹，里面有编译生成的各种用户空间的文件。可以把此目录下的各文件目录复制到 AM335X 的文件系统根目录下，同时内核模块 ethercat-1.5.1/devices/ec_generic.ko 和 ethercat-1.5.1/master/ec_master.ko 也需要手动复制到 AM335X 文件系统任意目录中。

4 EtherCAT测试

本文作者使用 AM335X BeagleBone 作为主站测试硬件，从站使用 AM335X 的 ICE 板。软件上主站使用上节编译出来的 IGH EtherCAT Master，从站使用 TI 提供的 Simple Demo Application。

首先通过 CCS 和 JTAG 仿真器连接 ICE 板，通过 JTAG 在 ICE 上面跑 EtherCAT Slave Simple Demo Application,并把 ICE 板和 BeagleBone 通过网线直连。

本例是通过主站每秒更新一次从站的 LED 的数据，并打印从站发过来的 switch 开关数字量。

确保硬件连接正确，而且在 ICE 板上已经成功运行从站的基础上使用串口终端连接 BeagleBone 板。启动 BeagleBone 板后顺序输入如下命令。

1) /etc/init.d/ethercat start

2) insmod output/modules/ec_generic.ko

上面两步加载了 EtherCAT 的内核模块，下面就可以使用 ethercat 工具来进行一些操作。

3) ethercat pdos

输入上面这条命令可以看到如下提示，说明软硬件正常。

```
SM0: PhysAddr 0x1000, DefaultSize  128, ControlRegister 0x26, Enable 1
SM1: PhysAddr 0x1400, DefaultSize  128, ControlRegister 0x22, Enable 1
SM2: PhysAddr 0x1800, DefaultSize    4, ControlRegister 0x64, Enable 1
  RxPDO 0x1601 "RxPDO-Map"
    PDO entry 0x7010:00, 32 bit, "SubIndex 000"
SM3: PhysAddr 0x1c00, DefaultSize    4, ControlRegister 0x20, Enable 1
  TxPDO 0x1a00 "TxPDO-Map"
    PDO entry 0x6000:00, 32 bit, "SubIndex 000"
```

每个 EtherCAT 从站控制器都有 64KByte 的本地地址空间(在 PRU 内部实现)，被划分为寄存器和双口 RAM 两部分。前 4KByte 是寄存器地址空间，双口 RAM 从地址 0x1000 开始。双口 RAM 的大小取决于具体实现，最大支持 60KByte。

上述信息表明该设备拥有 4 个同步管理通道。前两个为邮箱传输方式，用于 COE(CAN Over EtherCAT)协议的通讯，负责对 SDO 的传输。SDO 是 COE 协议的非周期数据传输方式(邮箱传输方式)。通过 SDO 可实现参数的设置与读取。为了能够在周期任务程序中直接对某个参数进行操作，应用程序需要在配置阶段创建一个针对该参数的 SDO 请求，并设置请求超时的时间。后两个为过程数据传输方式，负责对 PDO 的传输。其中 PhysAddr 参数为物理起始地址，即该同步管理通道在双口 RAM 上的起始地址；ControlRegister 参数为控制字，包含了该通道的传输方式、传输方向等信息。DefaultSize 规定了该通道的大小。第 4 行起开始定义 RxPdo，与 CANopen 协议类似，EtherCAT 的 PDO 也通过索引号(0x7010 和 0x6000)和子索引号（两个都是 0x00）进行识别。将 0x1601 定义为 RxPdo(接收 PDO)，在这个接收 PDO 中安排被主站接收的参数。将 0x1A00 定义为 TxPdo(发送 PDO)，在这个发送 PDO 中安排向主站发送的参数。

4) 执行测试程序”./ethercat_test”

此时观察 ICE 板可以看到 LED 灯在计数闪烁，同时通关短接 ICE 板 J9 的某些开关可以看到串口终端打印开关量也在变化。

5 EtherCAT测试程序分析

笔者针对 TI 的 EtherCAT 从站 demo 开发了基于 IGH EtherCAT 的对应主站测试例程（ethercat_test.c），此测试程序参考了 ethercat-1.5.1\examples\user 工程，下面详细介绍此程序。

1) 首先定义从站的 pdo 信息，这里根据命令“ethercat pdos”反馈的信息填充相关数据结构：

```
ec_pdo_entry_info_t slave_0_pdo_entries[] = {
    {0x7010, 0x00, 32}, /* SubIndex 000 */
    {0x6000, 0x00, 32}, /* SubIndex 000 */
};

ec_pdo_info_t slave_0_pdos[] = {
    {0x1601, 1, slave_0_pdo_entries + 0}, /* RxPDO-Map */
    {0x1a00, 1, slave_0_pdo_entries + 1}, /* TxPDO-Map */
};
```

在 main 函数里，先申请 master

```
    master = ecrt_request_master(0);
    if (!master)
        return -1;
```

创建两个过程数据域

```
    domain1 = ecrt_master_create_domain(master);
    if (!domain1)
        return -1;
    domain2 = ecrt_master_create_domain(master);
    if (!domain2)
        return -1;
```

获取从站的配置

```
  sc = ecrt_master_slave_config(master, BusCouplerPos, TI_AM3359ICE);
    if (!sc) {
        fprintf(stderr, "Failed to get slave configuration.\n");
        return -1;
    }
```

注册 PDO entry 到过程数据域中：

```
    off_dig_out2 = ecrt_slave_config_reg_pdo_entry(sc,
            0x7010, 0, domain1, NULL);
    if (off_dig_out2 < 0)
        return -1;
   off_dig_in2 = ecrt_slave_config_reg_pdo_entry(sc,
            0x6000, 0, domain2, NULL);
    if (off_dig_in2 < 0)
        return -1;
```

结束配置，开始周期性的任务

```
    if (ecrt_master_activate(master))
        return -1;
```

本例的任务的实时性是依靠 linux timer 来实现的。首先申请信号处理函数

```
    sa.sa_handler = signal_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGALRM, &sa, 0)) {
        fprintf(stderr, "Failed to install signal handler!\n");
        return -1;
    }
```

创建一个实时的 timer,这种间隔定时器在启动后，不管进程是否运行，每个时钟滴答都将其间隔计数器减 1。当减到 0 值时，内核向进程发送 SIGALRM 信号。虽然，在内核中间隔定时器的间隔计数器是以时钟滴答次数为单位，但是让用户以时钟滴答为单位来指定间隔定时器的间隔计数器的初值显然是不太方便的，因为用户习惯的时间单位是秒、毫秒或微秒等。所以 Linux 定义了数据结构 itimerval 来让用户以秒或微秒为单位指定间隔定时器的时间间隔值。it_interval 成员表示间隔计数器的初始值，而 it_value 成员表示间隔计数器的当前值。

```
    tv.it_interval.tv_sec = 0;
    tv.it_interval.tv_usec = 1000000 / FREQUENCY;
    tv.it_value.tv_sec = 0;
    tv.it_value.tv_usec = 1000;
    if (setitimer(ITIMER_REAL, &tv, NULL)) {
        fprintf(stderr, "Failed to start timer: %s\n", strerror(errno));
        return 1;
     }
```

在 while()循环和信号处理函数配合可以保证每收到一个 ALARM 信号执行一次 cyclic_task()函数。下面分析 cyclic_task()函数，这个函数每十毫秒调用一次。

首先，接收每个 domain 的数据并检查 domain 的状态：

```
    ecrt_master_receive(master);
    ecrt_domain_process(domain1);
    ecrt_domain_process(domain2);
    check_domain1_state();
    check_domain2_state();
```

每一秒更新一次 LED 数据，并打印开关数字量：

```
    if (counter) {
        counter--;
    } else { // do this at 1 Hz
        counter = FREQUENCY;
        // calculate new process data
        blink ++;
        // check for master state (optional)
        check_master_state();
                    printf("AnaIn: value 0x%x\n", EC_READ_U32(domain2_pd +
off_dig_in2));
    }
```

周期认为函数结束的时候写通过 EC_WRITE_U32()函数接口把数据写到结构体中，并通过调用函数 ecrt_domain_queue()和 ecrt_master_send()发送 process data:

```
    EC_WRITE_U32(domain1_pd + off_dig_out2, blink);
    ecrt_domain_queue(domain1);
    ecrt_domain_queue(domain2);
    ecrt_master_send(master);
```

至此 ethercat_test 程序分析完毕，需要使用如下命令编译：

```
arm-none-linux-gnueabi-gcc ethercat_test.c -o ethercat_test -I../ethercat-
1.5.1/output/include/  -L../ethercat-1.5.1/output/lib -lethercat
```

编译完成后把生成的二进制可执行文件复制到 BeagleBone 的文件系统任意目录中并执行之。

6 结束语

本文首先介绍了 EtherCAT 的基本原理和 TI 对 EtherCAT 的软硬件支持资源以及运行 TI 从站范例的步骤和需要注意的问题。然后介绍开源的 IGH EtherCAT 主站软件，详细介绍了如何把 IGH EtherCAT Master 移植到基于 AM335X 的 BeagleBone 上面。此外笔者还配合 TI 的 EtherCAT 从站 demo 开发了基于 IGH EtherCAT 的主站测试例程，详细分析了该例程的实现。

7 参考文献

1. *http://www.ethercat.org/*
2. *http://processors.wiki.ti.com/index.php/Configuring_TwinCAT_For_AM335x*
3. *http://etherlab.org/en/ethercat/index.php*
4. *马春敏;基于 Linux 的 EtherCAT 主站的研究;制造业自动化;2011 年 08 期*