Rockchip EtherCAT IgH 开发文档

文件标识：RK-YH-YF-A18

发布版本：V1.0.2

日期：2024-08-31

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

产品版本

芯片名称	内核版本
RK3588	Linux5.10
RK3576	Linux6.1
RK3568	Linux5.10 / Linux4.19
RK3506	Linux6.1

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	Zack.Huang	2024-03-11	初始版本
V1.0.1	Zack.Huang	2024-07-16	增加RK3506性能数据，修改软件环境说明
V1.0.2	Zack.Huang	2024-08-31	增加RK3506内核配置

目录

1. 概述

IgH EtherCAT为运行于Linux系统的开源EtherCAT主站程序，IgH EtherCAT主站通过构建Linux字符设备，应用程序通过对字符设备的访问实现与EtherCAT主站模块的通信。

IgH EtherCAT开发包配套EtherCAT工具，该工具提供各种可在Linux用户层运行的命令，可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。

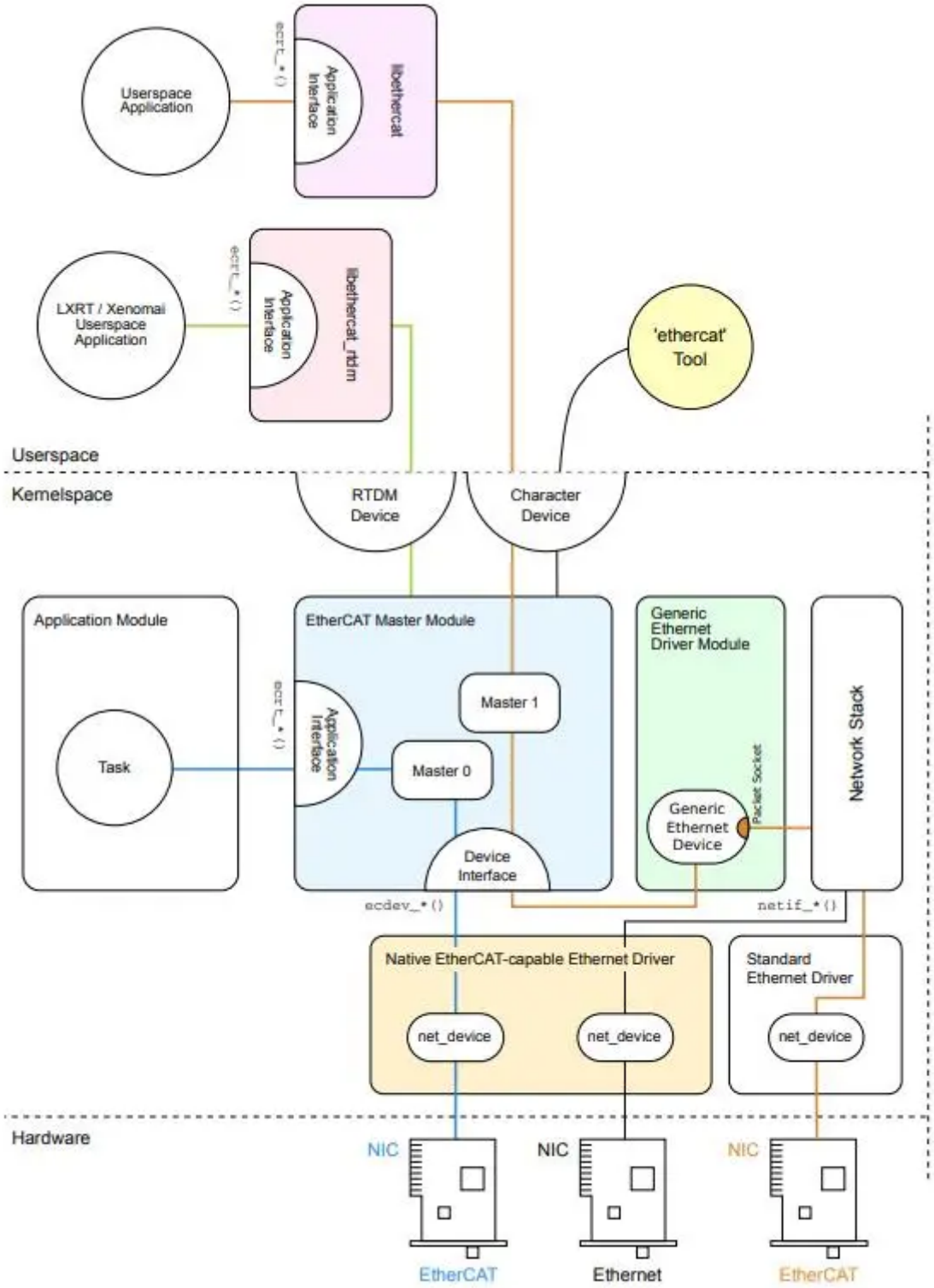


Figure 2.1: Master Architecture

具体详情可以查看IgH的官网：https://docs.etherlab.org/ethercat/1.5/doxygen/index.html

2. EtherCAT IgH软件环境基本介绍

EtherCAT整体分为四个部分，内核，驱动部分，用户态部分还有EtherCAT应用部分。

我们将提供预编译的bin文件：

	是否为预编译	包含的文件	作用
驱动部分	Y	phylink.ko，pcs-xpcs.ko，ec_master.ko，ec_stmmac.ko	提供实时性驱动环境
用户态部分	Y	ethercat，libethercat.so	ethercat是EtherCAT IgH的调试工具，libethercat.so提供用户层应用的调用接口
EtherCAT应用部分	N	libmadht1505ba1.so，rk_test，etc.	上层应用

2.1 内核

EtherCAT IgH需要保证高实时性，Preempt-RT是一种针对实时性能进行了优化的Linux内核。与普通的Linux内核相比，Preempt-RT具有以下优势：

1. 实时性能： Preempt-RT提供了更可靠和更精确的实时性能。它采用了一些实时调度策略和机制，使得任务能够按照严格的时间要求执行，从而适用于需要高度可预测性和低延迟的应用场景，如工业自动化、机器人控制等。
2. 硬实时能力： Preempt-RT具有硬实时能力，即能够确保任务在规定的时间内完成，而不会受到其他任务或中断的干扰。这对于需要严格的时间限制的应用非常重要，如航空航天、医疗设备等领域。
3. 任务调度： Preempt-RT使用了更加高效和优化的任务调度算法，如基于优先级的实时调度算法，以确保高优先级任务能够及时响应并完成，而低优先级任务不会影响到实时任务的执行。
4. 中断处理： Preempt-RT针对中断处理进行了优化，使得中断的响应时间更短，能够更快地响应外部事件。
5. 内核定时器： Preempt-RT提供了更精确和可配置的内核定时器，使得可以实现微秒级的定时精度，适用于对时间要求极高的应用场景。
6. 实时扩展： Preempt-RT提供了一些实时扩展机制，使得用户能够方便地对内核进行定制和扩展，以满足特定应用的需求。

总的来说，Preempt-RT在实时性能、可靠性和精度方面比普通的Linux内核更加优秀，因此在对实时性能要求较高的应用场景中被广泛使用。瑞芯微提供配套SDK的Preempt-RT补丁。

2.2 用户态部分

用户态中主要是两个文件，ethercat和libethercat.so，一个是EtherCAT IgH的调试工具，一个是EtherCAT IgH的动态库，用来提供用户层接口。用户态部分访问https://gitlab.com/etherlab.org/ethercat/-/tree/master?ref_type=heads 下载IgH的源代码，编译后会生成ethercat二进制文件，libethercat.so和一些example等等。用户态部分也可以使用预编译好的EtherCAT IgH二进制文件和libethercat.so。

用户态部分	具体作用	存放路径
libethercat.so	对ethercat 标准的实现，提供给上层应用提供接口	SDK/external/rk_ethercat_release/ethercat_igh/lib/

用户态部分	具体作用	存放路径
ethercat	ethercat是EtherCAT IgH的调试工具	SDK/external/rk_ethercat_release/ethercat_igh/bin/

2.3 驱动部分

驱动部分主要是ec_master.ko和一些RK优化后的ko。

驱动部分	具体作用	存放路径
ec_master.ko	ec_master.ko 是一个 Linux 内核模块，用于支持 EtherCAT 主站的功能。这个模块负责管理 EtherCAT 总线上的通信，实现主站与从站之间的数据交换和同步	SDK/external/rk_ethercat_release/driver
ec_stmmac.ko	stmmac是RK对底层网口驱动的实时性优化过的网口驱动	SDK/external/rk_ethercat_release/driver
pcs-xpcs.ko	用于支持 EtherCAT 网卡的功能	SDK/external/rk_ethercat_release/driver
phylink.ko	用于管理网络接口的物理层和链路层之间的通信，以及与物理介质相关的设置和状态管理	SDK/external/rk_ethercat_release/driver

2.4 EtherCAT IgH应用部分

EtherCAT IgH应用部分需要根据实际使用的伺服驱动器来实现，该部分无法通用的，RK提供的参考示例，使用的是如下配套组件：

组件	名称
主站	RK3588/RK3576/RK3568 开发板
从站驱动器	松下MADHT1505BA1
伺服电机	松下MDMS012S1U
连接线	动力线和编码线

3. EtherCAT IgH软件编译和部署

3.1 内核

3.1.1 步骤一：打上Preempt-RT补丁

内核方案选择Preempt-RT内核方案，参考文档打上补丁：

SDK/docs/Patches/Real-Time-Performance/README.md

要特别注意的是：发布的驱动部分SDK/external/rk_ethercat_release和内核头文件强相关，请确认内核的commit要和SDK/external/rk_ethercat_release/driver/redmine中描述的一样，再打入Preempt-RT补丁。

3.1.2 步骤二：开启对应内核配置

3.1.2.1 5.10内核（以RK3568举例）

```
#修改内核配置 把stmmac驱动设置为模块编译
--- a/arch/arm64/configs/rockchip_linux_defconfig
+++ b/arch/arm64/configs/rockchip_linux_defconfig
@@ -202,7 +202,7 @@ CONFIG_R8168=y
 # CONFIG_NET_VENDOR_SILAN is not set
 # CONFIG_NET_VENDOR_SIS is not set
 # CONFIG_NET_VENDOR_SMSC is not set
-CONFIG_STMMAC_ETH=y
+CONFIG_STMMAC_ETH=m
 # CONFIG_NET_VENDOR_SUN is not set
 # CONFIG_NET_VENDOR_SYNOPSYS is not set
 # CONFIG_NET_VENDOR_TEHUTI is not set

#在dts中disable另一个网口
--- a/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10-linux.dts
+++ b/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10-linux.dts
@@ -15,3 +15,7 @@
 &vp1 {
        cursor-win-id = <ROCKCHIP_VOP2_CLUSTER1>;
 };
+
+&gmac0 {
+       status = "disabled";
+};

#把cpu休眠关闭（这里也可以通过关闭CONFIG_ARM_PSCI_CPUIDLE来阻止cpu休眠）
--- a/arch/arm64/boot/dts/rockchip/rk3568.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568.dtsi
index 778752440303..dba79303e34b 100644
--- a/arch/arm64/boot/dts/rockchip/rk3568.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568.dtsi
@@ -74,7 +74,7 @@
                        enable-method = "psci";
                        clocks = <&scmi_clk 0>;
                        operating-points-v2 = <&cpu0_opp_table>;
-                       cpu-idle-states = <&CPU_SLEEP>;
+                       //cpu-idle-states = <&CPU_SLEEP>;
```

```
                        #cooling-cells = <2>;
                        dynamic-power-coefficient = <187>;
                };
@@ -86,7 +86,7 @@
                        enable-method = "psci";
                        clocks = <&scmi_clk 0>;
                        operating-points-v2 = <&cpu0_opp_table>;
-                       cpu-idle-states = <&CPU_SLEEP>;
+                       //cpu-idle-states = <&CPU_SLEEP>;
                };

                cpu2: cpu@200 {
@@ -96,7 +96,7 @@
                        enable-method = "psci";
                        clocks = <&scmi_clk 0>;
                        operating-points-v2 = <&cpu0_opp_table>;
-                       cpu-idle-states = <&CPU_SLEEP>;
+                       //cpu-idle-states = <&CPU_SLEEP>;
                };

                cpu3: cpu@300 {
@@ -106,7 +106,7 @@
                        enable-method = "psci";
                        clocks = <&scmi_clk 0>;
                        operating-points-v2 = <&cpu0_opp_table>;
-                       cpu-idle-states = <&CPU_SLEEP>;
+                       //cpu-idle-states = <&CPU_SLEEP>;
                };

                idle-states {
```

#隔离cpu核心

```
--- a/arch/arm64/boot/dts/rockchip/rk3568-linux.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568-linux.dtsi
@@ -13,7 +13,7 @@
        };

        chosen: chosen {
-               bootargs = "earlycon=uart8250,mmio32,0xfe660000 console=ttyFIQ0
root=PARTUUID=614e0000-0000 rw rootwait";
+               bootargs = "earlycon=uart8250,mmio32,0xfe660000 console=ttyFIQ0
root=PARTUUID=614e0000-0000 isolcpus=3 nohz_full=3 rw rootwait";
        };

        fiq-debugger {
```

```
diff --git a/arch/arm64/configs/rockchip_linux_defconfig
b/arch/arm64/configs/rockchip_linux_defconfig
index 1a342fe17a5d..1d3a939659b1 100644
--- a/arch/arm64/configs/rockchip_linux_defconfig
+++ b/arch/arm64/configs/rockchip_linux_defconfig
@@ -649,3 +649,4 @@ CONFIG_RCU_CPU_STALL_TIMEOUT=60
 CONFIG_FUNCTION_TRACER=y
 CONFIG_BLK_DEV_IO_TRACE=y
 CONFIG_LKDTM=y
+CONFIG_NO_HZ_FULL=y
```

3.1.2.2 6.1内核（以RK3576举例）

```
#修改内核配置 把stmmac驱动设置为模块编译：
--- a/arch/arm64/configs/rockchip_linux_defconfig
+++ b/arch/arm64/configs/rockchip_linux_defconfig
@@ -197,7 +197,7 @@ CONFIG_R8168=y
 # CONFIG_NET_VENDOR_SILAN is not set
 # CONFIG_NET_VENDOR_SIS is not set
 # CONFIG_NET_VENDOR_SMSC is not set
-CONFIG_STMMAC_ETH=y
+CONFIG_STMMAC_ETH=m
 # CONFIG_NET_VENDOR_SUN is not set
 # CONFIG_NET_VENDOR_SYNOPSYS is not set
 # CONFIG_NET_VENDOR_TEHUTI is not set

 --- a/arch/arm64/configs/rockchip_rt.config
+++ b/arch/arm64/configs/rockchip_rt.config
@@ -22,3 +22,4 @@ CONFIG_JUMP_LABEL=y
 CONFIG_HZ_PERIODIC=y
 CONFIG_HZ_1000=y
 CONFIG_PREEMPT_RT=y
+# CONFIG_DEBUG_PREEMPT is not set

 #关闭另一个gmac1网口
 --- a/arch/arm64/boot/dts/rockchip/rk3576-evb1-v10-linux.dts
+++ b/arch/arm64/boot/dts/rockchip/rk3576-evb1-v10-linux.dts
@@ -14,3 +14,7 @@
         model = "Rockchip RK3576 EVB1 V10 Board";
         compatible = "rockchip,rk3576-evb1-v10", "rockchip,rk3576";
 };
+
+&gmac1 {
+       status = "disabled";
+};

#隔离cpu
--- a/arch/arm64/boot/dts/rockchip/rk3576-linux.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3576-linux.dtsi
@@ -6,7 +6,7 @@

 / {
         chosen: chosen {
-               bootargs = "earlycon=uart8250,mmio32,0x2ad40000 console=ttyFIQ0
clk_gate.always_on=1 pm_domains.always_on=1 root=PARTUUID=614e0000-0000 rw
rootwait rcupdate.rcu_expedited=1 rcu_nocbs=all";
+               bootargs = "earlycon=uart8250,mmio32,0x2ad40000 console=ttyFIQ0
clk_gate.always_on=1 pm_domains.always_on=1 root=PARTUUID=614e0000-0000
isolcpus=7 nohz_full=7 rw rootwait rcupdate.rcu_expedited=1 rcu_nocbs=all";
         };

         fiq_debugger: fiq-debugger {
```

3.1.2.3 6.1内核（RK3506）

```
diff --git a/arch/arm/configs/rk3506_defconfig
b/arch/arm/configs/rk3506_defconfig
index e6d3ba608a77..3029973ade8c 100644
--- a/arch/arm/configs/rk3506_defconfig
+++ b/arch/arm/configs/rk3506_defconfig
@@ -2,6 +2,7 @@
 CONFIG_KERNEL_XZ=y
 CONFIG_DEFAULT_HOSTNAME="localhost"
 CONFIG_SYSVIPC=y
+CONFIG_NO_HZ_FULL=y
 CONFIG_NO_HZ=y
 CONFIG_HIGH_RES_TIMERS=y
 CONFIG_PREEMPT=y
@@ -119,12 +120,13 @@ CONFIG_NETDEVICES=y
 # CONFIG_NET_VENDOR_SOLARFLARE is not set
 # CONFIG_NET_VENDOR_SMSC is not set
 # CONFIG_NET_VENDOR_SOCIONEXT is not set
-CONFIG_STMMAC_ETH=y
+CONFIG_STMMAC_ETH=m
 # CONFIG_DWMAC_GENERIC is not set
 # CONFIG_NET_VENDOR_SYNOPSYS is not set
 # CONFIG_NET_VENDOR_VIA is not set
 # CONFIG_NET_VENDOR_WIZNET is not set
 # CONFIG_NET_VENDOR_XILINX is not set
+CONFIG_PHYLIB=y
 CONFIG_MOTORCOMM_PHY=y
 CONFIG_PPP=y
 # CONFIG_WLAN is not set
```

3.1.3 步骤三：编译出实时内核

然后根据rt-linux README.md 生成实时性内核。

3.2 用户态部分

这部分可以直接在SDK中获取（参考第二章），或者源码编译，源码linux环境下编译的方法如下：

```
#进入源码目录
cd source

#指定编译链路径（编译链在SDK中有提供）
export PATH=SDK/prebuilts/gcc/linux-x86/(arch)/gcc-arm-10.3-2021.07-x86_64-
(arch)-none-linux-gnu/bin:$PATH

#配置,指定内核目录，选择SDK下的kernel-6.1或者kernel-5.10，这里使用kernel-6.1。
./bootstrap
```

```
./configure --prefix=（编译后你需要存放的目录） --host=(arch)-none-linux-gnu --with-linux-dir=SDK/kernel-6.1 --enable-8139too=no --enable-stmmac=yes --enable-generic=no --enable-wildcards=yes

#编译
make -j8
make install systemdsystemunitdir=（编译后你需要存放的目录，和configure命令上的prefix下跟的参数需要一直）

#部署，从你指定的输出目录中拷贝文件到开发板上
cp libethercat.so*  /usr/lib/(开发板上)
cp ethercat  /usr/bin/(开发板上)
```

3.3 驱动部分

```
cp phylink.ko /userdata/(开发板上)  内核编译
cp pcs-xpcs.ko /userdata/(开发板上) 内核编译
cp ec_master.ko /userdata/(开发板上)
cp ec_stmmac.ko /userdata/(开发板上)

#加载驱动
insmod /userdata/phylink.ko
insmod /userdata/pcs-xpcs.ko
insmod /userdata/ec_master.ko main_devices=62:36:B8:01:5B:59（这个是你的网口的物理地址，可以通过ifconfig命令查看）
insmod /userdata/ethercat_out/ec_stmmac.ko

#调整soc为性能模式
echo performance | tee $(find /sys/ -name *governor)
```

3.4 EtherCAT IgH应用部分

EtherCAT IgH应用部分需要根据具体的伺服器来编写，需要使用上面用户态编译出来的EtherCAT IgH工具获取一些伺服驱动器的配置信息，然后完成应用代码的开发。详情请见下面 ”EtherCAT IgH应用部分开发指导“ 章节。

4. EtherCAT IgH工具介绍

使用ethercat --help命令可以查看EtherCAT IgH工具，这些工具可以查看与主站连接的从站的各种信息，可以有助于主站应用程序的编写，下面介绍几种常用的命令和参数的使用，其中[]中为必选参数，<>为可选参数。

```
#设置别名地址(下面的例子代表设置位置0的别名修改为1)
ethercat alias -p 0 0

#设置别名地址(下面的例子代表别名0的别名修改为1)
ethercat alias -a 0 1
```

```
#以c语言的形式输出pdo信息
ethercat cstruct

#输出识别到的从站
ethercat slaves
0  0(别名)):0（位置）  PREOP  +  MADHT1505BA1

#输出pdo信息
ethercat pdos
```

还有的工具请详见官方文档https://docs.etherlab.org/ethercat/1.5/doxygen/index.html

5. EtherCAT IgH应用开发参考

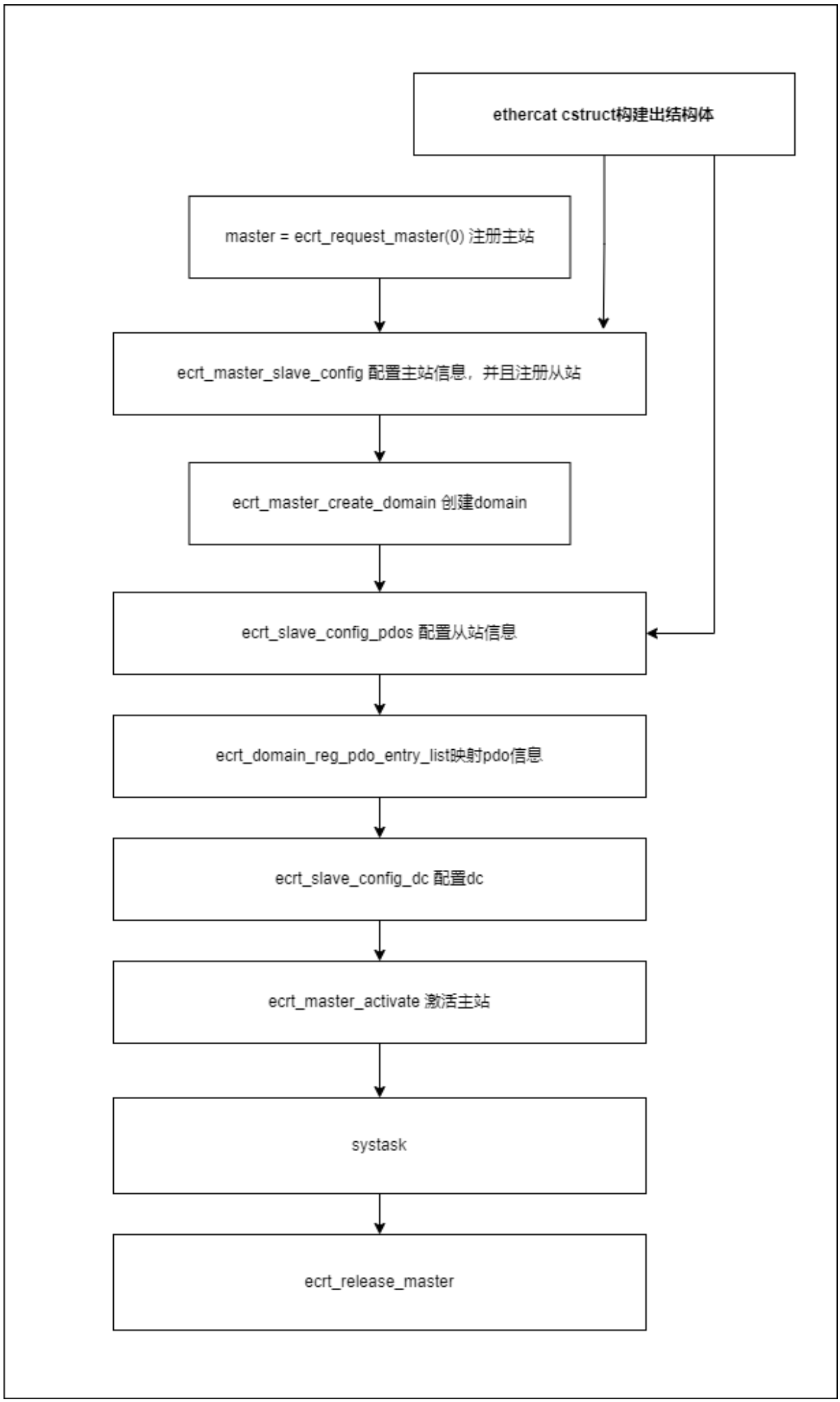
开发板主站和从站连接上电，伺服电机连接从站后，开发板上加载驱动后，使用以下命令查看是否通讯正常：

```
#输出识别到的从站
ethercat slaves

#如果通讯成功，就会显示出识别到的从站位置信息，别名和型号，如下：
0  0(别名)):0（位置）  PREOP  +  MADHT1505BA1

#输出pdo信息，这些pdo信息只是默认的，具体要根据需求，参考伺服驱动器手册来编写
ethercat pdos

#以c语言的形式输出pdo信息
ethercat cstruct
```

以上获取的基本信息，就可以编写一个简单的EtherCAT IgH应用了，可以参考ethercat_igh\examples\dc_user\main.c代码。

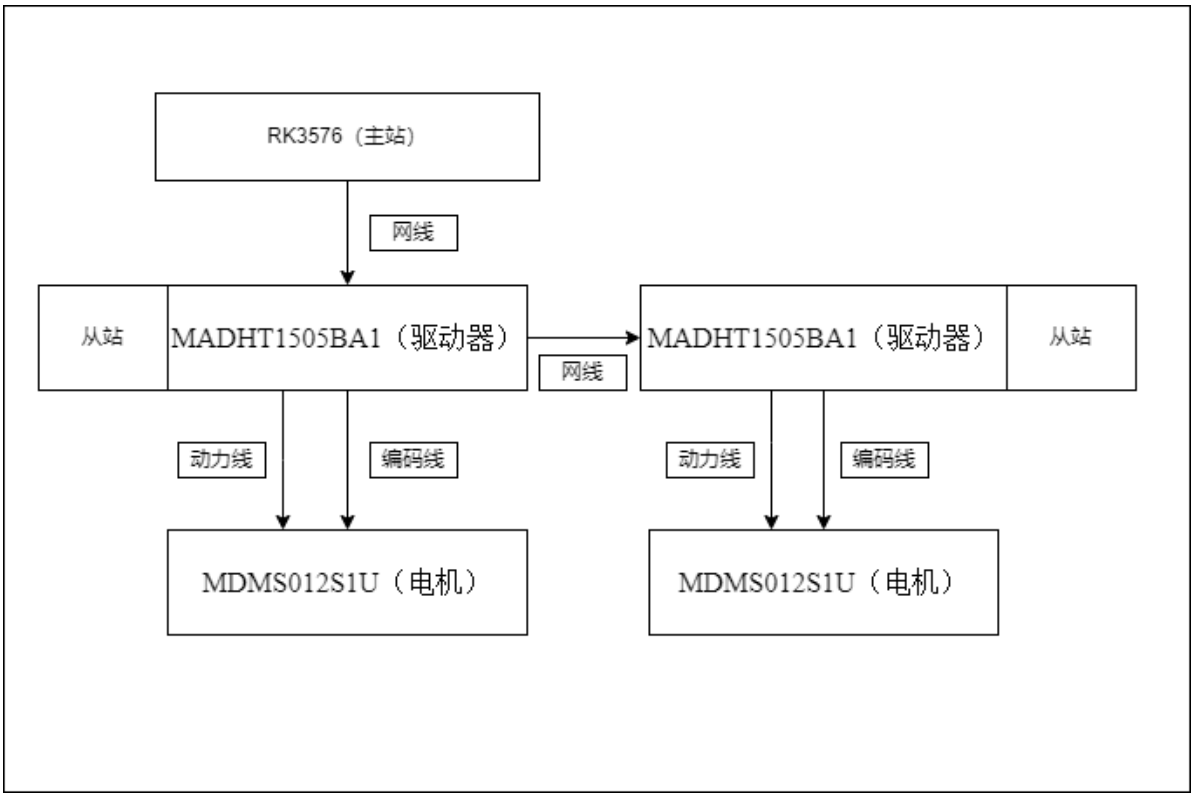
使用ethercat cstruct获取到的pdo信息替换官方例子中的el3102_pdo_entries和el4102_pdos等pdos信息，就可以简单做一个初始化.

主要流程如下：



这里提供一个RK自研样机的例子作为示例：

以下是RK自研样机的拓扑结构图：



主站通过网线连接一台MADHT1505BA1，MADHT1505BA1通过网线连接另一台MADHT1505BA1，电机分别使用动力线和编码线和驱动器MADHT1505BA1相连。

可以从下面获取RK自研样机的上层应用程序：

```
SDK/external/rk_ethercat_release/demo
```

可以作为编写客户自己的上层应用的参考。

我们还提供一些电机控制UI上的demo设计参考，和上面的电机测试程序配套使用。详见：

```
SDK/app/lvgl_demo/motor_demo
```

6. 性能指标

6.1 测试方法

参考代码SDK/external/rk_ethercat_release/demo/ec_master_test_RK3568_MADHT1505BA1.c中的cyclic_task_velocity_mode函数，里面有实现统计一个周期的所花费的最大值和最小值

```
static int clean_cycle = 0;//5 * 60 * FREQUENCY;
void cyclic_task_velocity_mode()
{
    static unsigned int timeout_error = 0;
    struct timespec wakeupTime, time;
    uint16_t    status;
    int8_t      opmode;
```

```
    int32_t     cur_velocity;
    bool        print = false;

    struct timespec startTime, endTime, lastStartTime = {};
    uint32_t period_ns = 0, exec_ns = 0, latency_ns = 0,
             latency_min_ns = 0, latency_max_ns = 0,
             period_min_ns = 0, period_max_ns = 0,
             exec_min_ns = 0, exec_max_ns = 0;

    period_max_ns = 0;
    period_min_ns = 0xffffffff;
    latency_max_ns = 0;
    latency_min_ns = 0xffffffff;

    clock_gettime(CLOCK_TO_USE, &lastStartTime);
    clock_gettime(CLOCK_TO_USE, &wakeupTime);
    while(app_run) {//这里会一直在循环
        wakeupTime = timespec_add(wakeupTime, cycletime);
        clock_nanosleep(CLOCK_TO_USE, TIMER_ABSTIME, &wakeupTime, NULL);

        // Write application time to master
        //
        // It is a good idea to use the target time (not the measured time) as
        // application time, because it is more stable.
        //
        ecrt_master_application_time(master, TIMESPEC2NS(wakeupTime));

        /*Receive process data*/
        ecrt_master_receive(master);
        ecrt_domain_process(domain);
        // check process data state (optional)
        check_domain_state();

        if (counter) {
            counter--;
        }else {
            counter = FREQUENCY;
            check_master_state();
            check_slave_config_states();
            /*Check process data state(optional)*/

            EC_WRITE_U16(domain_pd + control_word, 0x80);
            // EC_WRITE_U8(domain_pd + modes_of_operation, 9);
            /*Read inputs*/
            status = EC_READ_U16(domain_pd + status_word);
            opmode = EC_READ_U8(domain_pd + modes_of_operation_display);
            cur_velocity = EC_READ_S32(domain_pd + current_velocity);
            printf_debug("madht:  act velocity = %d ,  status = 0x%x, opmode =
0x%x\n", cur_velocity,  status, opmode);

            if( (status & 0x004f) == 0x0040) {
                printf_debug("0x06\n");
                EC_WRITE_U16(domain_pd + control_word, 0x0006);
                EC_WRITE_U8(domain_pd + modes_of_operation, 9);
            }
```

```
            else if( (status & 0x006f) == 0x0021) {
                printf_debug("0x07\n");
                EC_WRITE_U16(domain_pd + control_word, 0x0007);
            }

            else if( (status & 0x006f) == 0x0023) {
                printf_debug("0x0f\n");
                EC_WRITE_U16(domain_pd + control_word, 0x000f);
                EC_WRITE_S32(domain_pd + target_velocity, TARGET_VELOCITY);
            }

            //operation enabled
            else if( (status & 0x006f) == 0x0027) {
                printf_debug("0x1f\n");
                EC_WRITE_U16(domain_pd + control_word, 0x001f);
            }
        }

        clock_gettime(CLOCK_TO_USE, &time);
        ecrt_master_sync_reference_clock_to(master, TIMESPEC2NS(time));
        ecrt_master_sync_slave_clocks(master);
        // send process data
        ecrt_domain_queue(domain);
        ecrt_master_send(master);

        clock_gettime(CLOCK_TO_USE, &startTime);
        latency_ns = DIFF_NS(wakeupTime, startTime);
        period_ns = DIFF_NS(lastStartTime, startTime);
        //exec_ns = DIFF_NS(lastStartTime, endTime);
    /* clean */
        if (clean_cycle >= (5 * 60 * 1000)) {
            clean_cycle = 0;
            period_max_ns = 0;
            period_min_ns = 0xffffffff;
            latency_max_ns = 0;
            latency_min_ns = 0xffffffff;
        }

//计算抖动最大最小值
        if (latency_ns > latency_max_ns) {
            latency_max_ns = latency_ns;
        }
        if (latency_ns < latency_min_ns) {
            latency_min_ns = latency_ns;
        }

        if (period_ns > period_max_ns) {
            period_max_ns = period_ns;
            print = true;
        }
        if (period_ns < period_min_ns) {
            period_min_ns = period_ns;
            print = true;
        }
```

```
        clean_cycle++;
        lastStartTime = startTime;
        // output timing stats
        if (print) {
            printf("period     %10u ... %10u\n",
                    period_min_ns, period_max_ns);
        //printf("exec       %10u ... %10u\n",
        //        exec_min_ns, exec_max_ns);
        printf("latency    %10u ... %10u\n",
                latency_min_ns, latency_max_ns);
        }
        print = false;
    }
}
```

假设是周期是1ms，最大抖动就是最大值减去最小值，最大抖动可以评估主站的实时性。

6.2 测试数据

在1ms控制周期周期内，各个支持平台的性能数据如下：

SOC	周期	抖动延时
RK3588	1ms	15us
RK3576	1ms	30us
RK3568	1ms	50us
RK3506	1ms	100us