Rockchip EtherCAT IgH Developer Document

ID: RK-YH-YF-A18

Release Version: V1.0.2

Release Date: 2024-08-31

Security Level: ☐Top-Secret ☐Secret ☐Internal ■Public

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Product Version

SoC	Kernel Version
RK3588	Linux5.10
RK3576	Linux6.1
RK3568	Linux5.10 / Linux4.19
RK3506	Linux6.1

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Version	Author	Date	Change Description
V1.0.0	Zack.Huang	2024-03-11	Initial version
V1.0.1	Zack.Huang	2024-07-16	Added RK3506 performance data, modified software environment description
V1.0.2	Zack.Huang	2024-08-31	Added RK3506 kernel configuration

Contents

1. Overview

IgH EtherCAT is an open-source EtherCAT master station designed to run on Linux systems. The IgH EtherCAT master communicates with EtherCAT slave devices by constructing a Linux character device, allowing applications to access the EtherCAT master module through the character device.

The IgH EtherCAT development package includes EtherCAT tools, which provide various commands that can be executed at the Linux application level. These commands enable direct access to and configuration of slave devices, such as setting slave addresses, displaying bus configurations, displaying PDO data, and reading and writing SDO parameters.

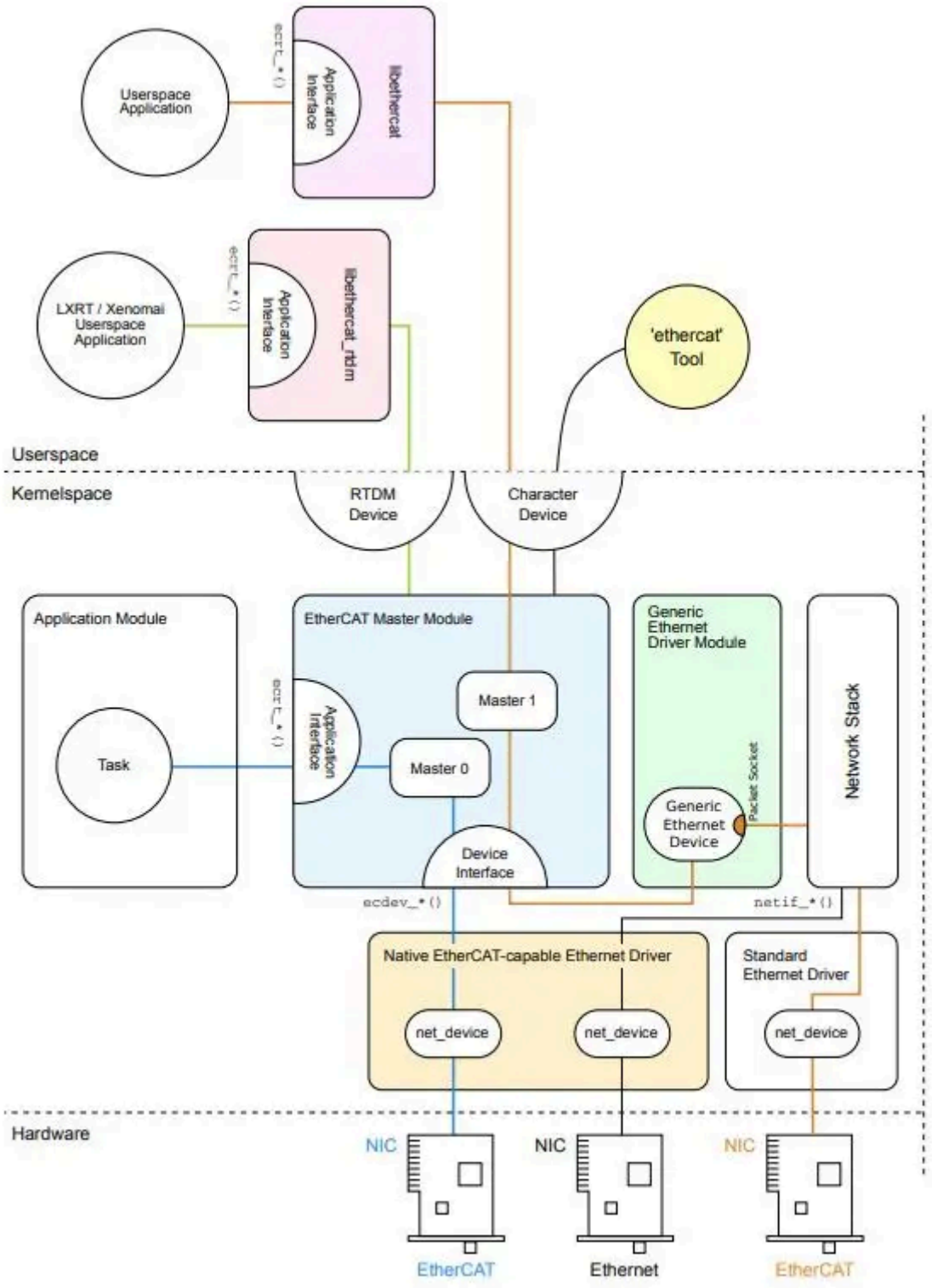


Figure 2.1: Master Architecture

For more details, please refer to the official website of IgH: [IgH EtherCAT Documentation]

2. Introduction to EtherCAT IgH Software Environment

EtherCAT is generally divided into four parts: kernel, driver, user-space, and EtherCAT application.

We will provide pre-compiled bin files:

	Is Pre-compiled	Included Files	Function
Driver	Y	phylink.ko, pcs-xpcs.ko, ec_master.ko, ec_stmmac.ko	Provide a real-time driver environment
User-Space	Y	ethercat, libethercat.so	ethercat is the debugging tool for EtherCAT IgH, libethercat.so provides the calling interface for user-level applications
EtherCAT Application	N	libmadht1505ba1.so, rk_test, etc.	Upper-layer applications

2.1 Kernel

EtherCAT IgH requires high real-time performance, and Preempt-RT is a Linux kernel optimized for real-time performance. Compared to the standard Linux kernel, Preempt-RT has the following advantages:

1. **Real-time performance:** Preempt-RT offers more reliable and precise real-time performance. It employs real-time scheduling strategies and mechanisms that allow tasks to be executed according to strict time requirements, making it suitable for applications that require high predictability and low latency, such as industrial automation and robotic control.
2. **Hard real-time capability:** Preempt-RT has hard real-time capabilities, meaning it can ensure that tasks are completed within a specified time frame without interference from other tasks or interrupts. This is crucial for applications that require strict time constraints, such as aerospace and medical equipment.
3. **Task scheduling:** Preempt-RT uses more efficient and optimized task scheduling algorithms, such as priority-based real-time scheduling algorithms, to ensure that high-priority tasks are responded to and completed in a timely manner without being affected by low-priority tasks.
4. **Interrupt handling:** Preempt-RT has been optimized for interrupt handling, resulting in shorter interrupt response times and faster reaction to external events.
5. **Kernel timers:** Preempt-RT provides more accurate and configurable kernel timers, enabling microsecond-level timing precision suitable for applications with extremely high time requirements.
6. **Real-time extensions:** Preempt-RT offers real-time extension mechanisms that allow users to easily customize and extend the kernel to meet the requirements of specific applications.

In summary, Preempt-RT excels in real-time performance, reliability, and precision compared to the standard Linux kernel and is widely used in applications with high real-time performance requirements. Rockchip provides the Preempt-RT patch that comes with the SDK.

2.2 User Space

The user space primarily consists of two files, ethercat and libethercat.so. One is the EtherCAT IgH debugging tool, and the other is the EtherCAT IgH dynamic library, which provides an interface for the user level. About the user space, please refer the IgH source code from https://gitlab.com/etherlab.org/ethercat/-/tree/master?ref_type=heads, and after compilation, it generates the ethercat binary file, libethercat.so, and some examples, etc. The user space components can also use pre-compiled EtherCAT IgH binary files and libethercat.so.

User Space	Function	Storage Path
libethercat.so	Implementation of the EtherCAT standard, providing an interface to applications	SDK/external/rk_ethercat_release/ethercat_igh/lib/
ethercat	Ethercat is the EtherCAT IgH debugging tool	SDK/external/rk_ethercat_release/ethercat_igh/bin/

2.3 Driver

The driver primarily consists of ec_master.ko and several kernel modules optimized by RK.

Driver	Detailed Function	Storage Path
ec_master.ko	ec_master.ko is a Linux kernel module that supports the functionality of EtherCAT Master. This module is responsible for managing communication on the EtherCAT bus, implementing data exchange and synchronization between the master and slave stations.	SDK/external/rk_ethercat_release/driver
ec_stmmac.ko	stmmac is a real-time optimized network driver for the underlying network interface by RK.	SDK/external/rk_ethercat_release/driver
pcs-xpcs.ko	Used to support the functionality of EtherCAT network cards.	SDK/external/rk_ethercat_release/driver
phylink.ko	Used for managing communication between the physical layer and link layer of network interfaces, as well as settings and status management related to the physical medium.	SDK/external/rk_ethercat_release/driver

2.4 EtherCAT IgH Application

The EtherCAT IgH application needs to be implemented based on the actual servo drives used, and This part is not universal. The reference example provided by RK uses the following supporting components:

Component	Name
Master	RK3588/RK3576/RK3568 Development Boards
Slave Driver	Panasonic MADHT1505BA1
Servo Motor	Panasonic MDMS012S1U
Connection	Power and Encoder Cables

3. EtherCAT IgH Software Compilation and Deployment

3.1 Kernel

3.1.1 Step One: Add Preempt-RT Patch

Choose the Preempt-RT kernel solution and add the patch according to the reference document:

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

It is important to note that: The released driver in SDK/external/rk_ethercat_release is strongly related to the kernel header files. Please ensure that the kernel commit matches the description in SDK/external/rk_ethercat_release/driver/redmine before adding the Preempt-RT patch.

3.1.2 Step Two: Enable Corresponding Kernel Configuration

3.1.2.1 5.10 Kernel (Taking RK3568 as an Example)

```
# Modify kernel configuration to compile stmmac driver as a module
--- 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

# Disable another network port in dts
--- 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";
+};

# Disable CPU idle (CPU idle can also be prevented by disabling
CONFIG_ARM_PSCI_CPUIDLE)
--- 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 {

# Isolate CPU cores
--- 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 Kernel (Take RK3576 as an Example)

```
# Modify kernel configuration to compile stmmac driver as a module:
--- 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

 # Disable another gmac1 network port
 --- 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";
+};

# Isolate CPUs
--- 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 Kernel (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 Step Three: Compile the Real-Time Kernel

Then, generate the real-time kernel based on the rt-linux README.md.

3.2 User Space

This part can be directly obtained from the SDK (refer to Chapter 2), or compiled from source code. The method for compiling the source code in a Linux environment is as follows:

```
# Enter the source code directory
cd source

# Specify the path to the toolchain (the toolchain is provided in the SDK)
```

```
export PATH=SDK/prebuilts/gcc/linux-x86/(arch)/gcc-arm-10.3-2021.07-x86_64-(arch)-none-linux-gnu/bin:$PATH

# Configuration, specify the kernel directory, choose kernel-6.1 or kernel-5.10 under the SDK, it is kernel-6.1 here.
./bootstrap
./configure --prefix=(the directory where you want to store the compiled files) --host=(arch)-none-linux-gnu --with-linux-dir=SDK/kernel-6.1 --enable-8139too=no --enable-stmmac=yes --enable-generic=no --enable-wildcards=yes

# Compile
make -j8
make install systemdsystemunitdir=(the directory where you want to store the compiled files, it should be consistent with the parameter following the prefix in the configure command)

# Deployment, copy files from your specified output directory to the development board
cp libethercat.so*  /usr/lib/(on the development board)
cp ethercat  /usr/bin/(on the development board)
```

3.3 Driver

```
cp phylink.ko /userdata/[Development Board] Kernel Compilation
cp pcs-xpcs.ko /userdata/[Development Board] Kernel Compilation
cp ec_master.ko /userdata/[Development Board]
cp ec_stmmac.ko /userdata/[Development Board]

# Load drivers
insmod /userdata/phylink.ko
insmod /userdata/pcs-xpcs.ko
insmod /userdata/ec_master.ko main_devices=62:36:B8:01:5B:59 (This is the physical address of your network interface, which can be checking by the ifconfig command)
insmod /userdata/ethercat_out/ec_stmmac.ko

# Adjust SOC to performance mode
echo performance | tee $(find /sys/ -name *governor)
```

3.4 EtherCAT IgH Application

The EtherCAT IgH application needs to be written based on the specific server, requiring the use of the EtherCAT IgH tools compiled in user space to obtain some configuration information of the servo drivers, and then complete the development of the application code. For more details, please refer to the section "EtherCAT IgH Application Developer Guide" below.

4. EtherCAT IgH Tools Introduction

EtherCAT IgH tools can be checked by `ethercat --help` command, these tools can display various information about the slaves connected to the master station. These tools can help the development of master station applications. Below are several commonly used commands and parameters, where [ ] is required parameters and < > is optional parameters.

```
# Set alias address (the example below represents changing the alias of position
0 to 1)
ethercat alias -p 0 0

# Set alias address (the example below represents changing the alias of alias 0
to 1)
ethercat alias -a 0 1

# Output PDO information in C language format
ethercat cstruct

# Output recognized slaves
ethercat slaves
0  0(alias)):0 (position)  PREOP  +  MADHT1505BA1

# Output PDO information
ethercat pdos
```

For additional tools, please refer to the official documentation https://docs.etherlab.org/ethercat/1.5/doxygen/index.html.

5. EtherCAT IgH Application Development Reference

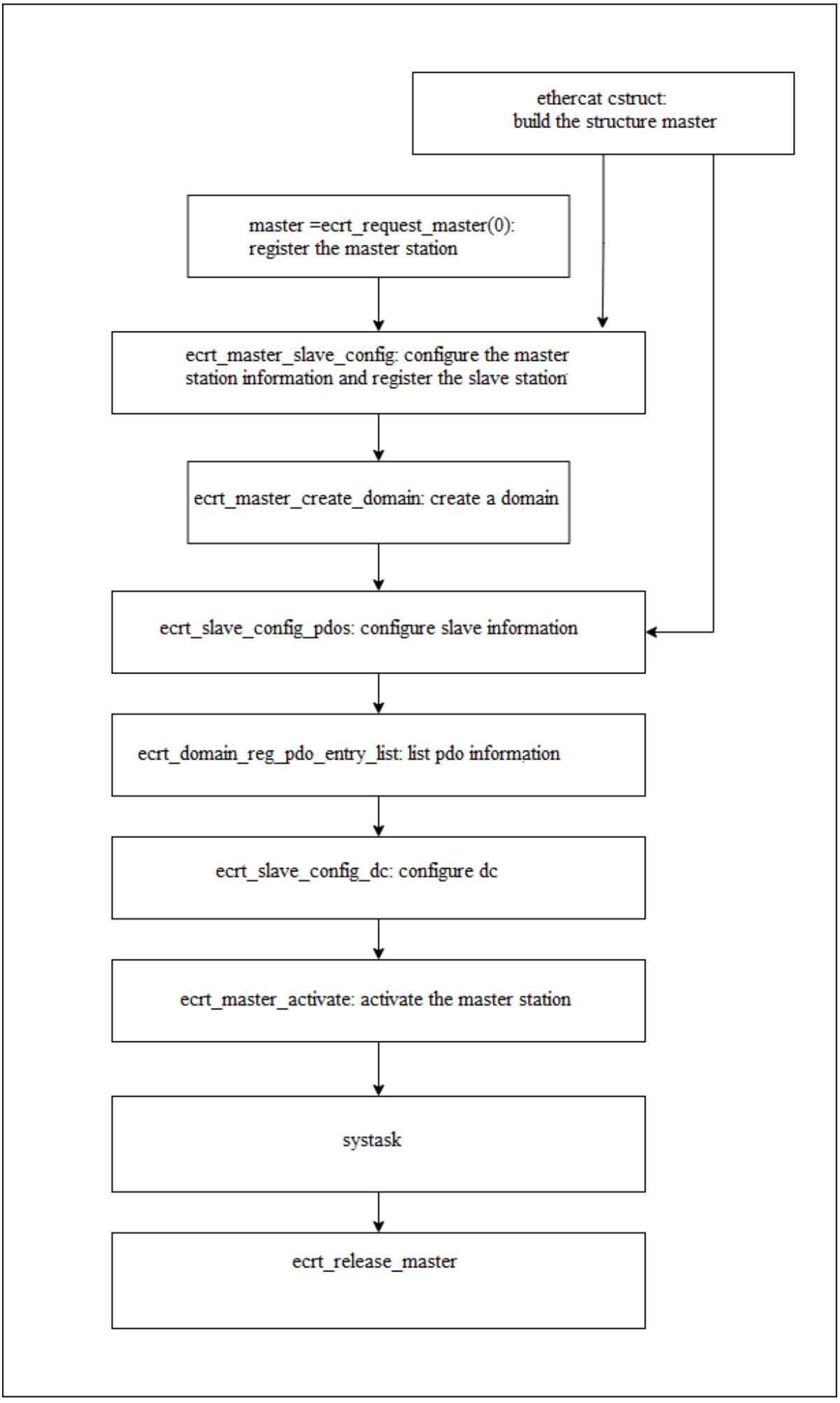
After powering up the master and slave stations of the development board, and connecting the servo motor to the slave station, load the driver on the development board and use the following commands to check if the communication is normal:

```
# Output recognized slaves
ethercat slaves

# If communication is successful, it will display the recognized slave position
information, alias, and model, as follows:
0  0(alias)):0(position)  PREOP  +  MADHT1505BA1

# Output PDO information, these PDO information are just default, and should be
written according to detailed requirements, please refer to the servo drive
manual
ethercat pdos

# Output PDO information in C language format
ethercat cstruct
```

With the basic information obtained above, you can write a simple EtherCAT IgH application, which can refer to the code in `ethercat_igh\examples\dc_user\main.c`.

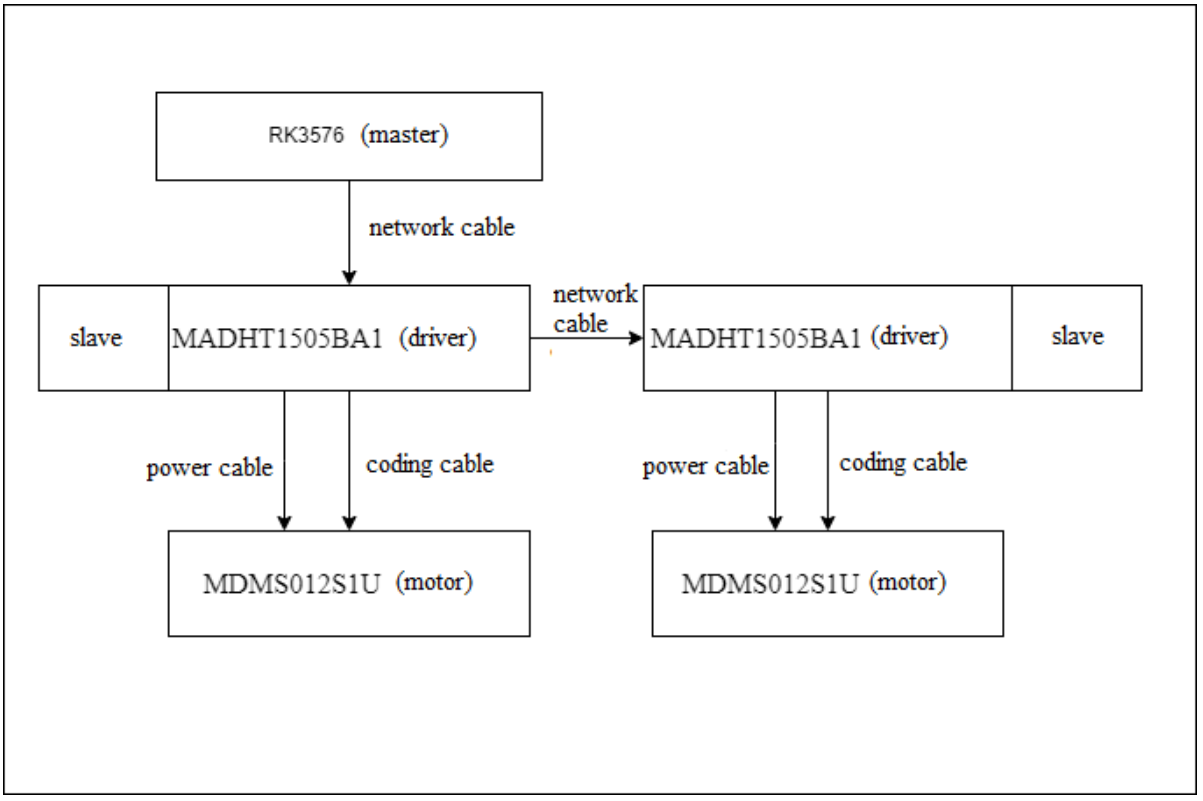
Replace the PDO information obtained by `ethercat cstruct` with the `el3102_pdo_entries` and `el4102_pdos` in the official example to perform a simple initialization.

The main process is as follows:



Here is an example of a self-developed demo by RK as a reference:

Below is the topology diagram of the RK self-developed demo:



The master station is connected to one MADHT1505BA1 via an Ethernet cable, and the MADHT1505BA1 is connected to another MADHT1505BA1 via an Ethernet cable, with the motors connected to the drive MADHT1505BA1 using power cables and encoder cables.

You can obtain the application of the RK self-developed demo from the following location:

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

This can be used as a reference for writing your own customize applications.

We also provide some demo designs for motor control UI, which can be used together with the above motor test programs. For details, please refer to:

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

6. Performance

6.1 Test Method

Please refer to the `cyclic_task_velocity_mode` function in the code SDK/external/rk_ethercat_release/demo/ec_master_test_RK3568_MADHT1505BA1.c, which implements the calculation of the maximum and minimum values spent in a cycle.

```
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) {//this loop will continue indefinitely
        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;
        }

//Calculate the maximum and minimum jitter
        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;
    }
}
```

Assuming the cycle is 1ms, the maximum jitter is the maximum value minus the minimum value, which can be used to assess the real-time performance of the master station.

6.2 Test Data

Within a 1ms control cycle, the performance data for each supported platform is as follows:

SOC	Cycle	Jitter Delay
Rockchip RK3588	1ms	15us
Rockchip RK3576	1ms	30us
Rockchip RK3568	1ms	50us
Rockchip RK3506	1ms	100us