AN1351: Using the Co-Processor Communication Daemon (CPCd)

This application note guides the user through the steps needed to properly configure and run the CPC daemon (CPCd) on Linux. It does not discuss how to use the CPC Library to write applications that interact with CPCd.

KEY POINTS

- Theory of operation
- Compiling and installing CPCd
- Configuring CPCd
- Troubleshooting
- Considerations

1 Introduction

Co-Processor Communication (CPC) enables one host system to communicate with a Network co-processor device (NCP), also named the secondary device or secondary, by physical transport (UART, SPI, and so on).

In CPC, data transfers between processors are segmented in sequential packets. Transfers are guaranteed to be error-free and sent in order. Multiple applications can send or receive on the same endpoint without worrying about collisions.

A CPC daemon (CPCd) is provided to allow applications on Linux to interact with a secondary running CPC.

The CPC daemon (CPCd) is distributed as three components:

- The daemon binary **(cpcd)**
- A library and associated header files that enable C applications to interact with the daemon (**libcpc.so**)
- A configuration file (**cpcd.conf**)

2 Theory of Operation

CPCd uses Unix sockets configured as sequential packets to transfer data with the Linux host applications. Data is then forwarded to the co-processor over a serial link. The Unix sockets, used to transfer data with applications that use the CPC Library (libcpc.so), are instantiated in the /tmp/cpcd folder.. A description of the CPC library usage is out of scope for this application note.

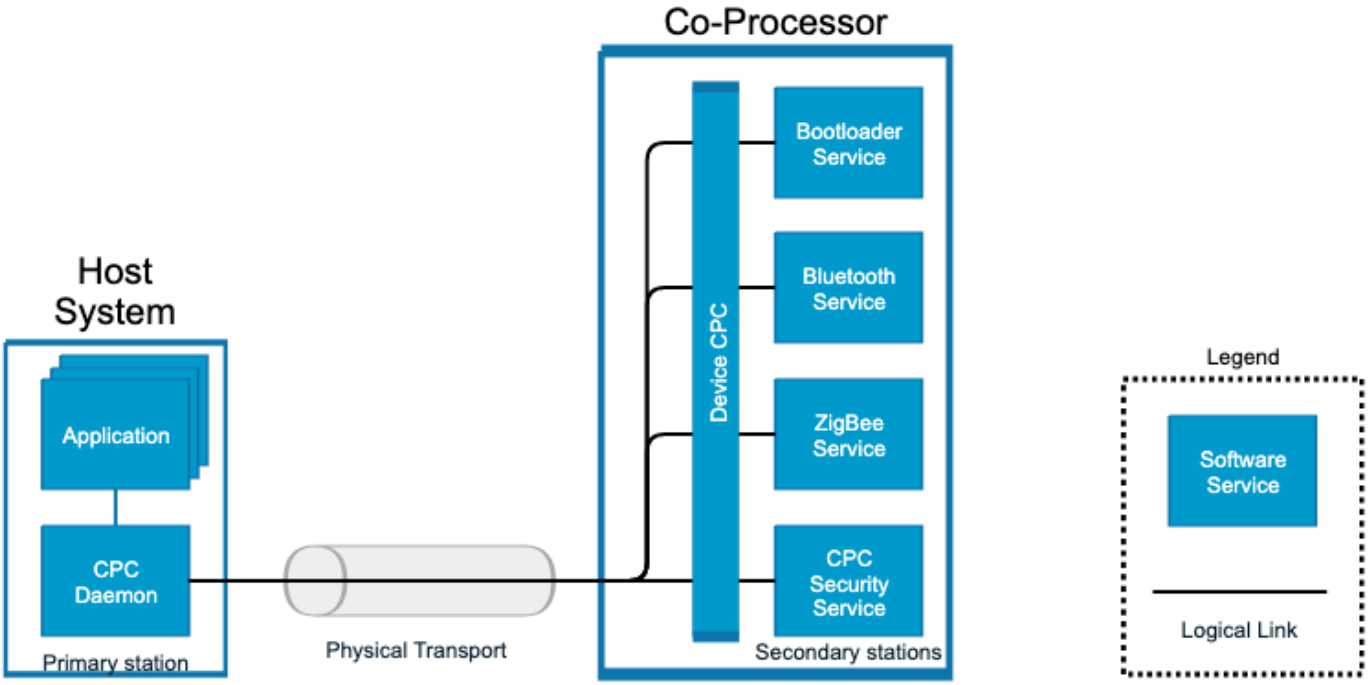

Figure 2.1. Co-Processor Communication Overview

3 Compiling, Installing, and Configuring CPCd

3.1 Downloading

Download the daemon source files from Silicon Labs GitHub project:

https://github.com/SiliconLabs/cpc_daemon

The main branch contains the latest official versions. Early access versions are available in the specific version branches.

3.2 Compiling CPCd and the CPC Library

The build essential and CMake packages in Linux are required for this step. Compile the CPC daemon in the source folder using the following commands:

```
mkdir build
cd build
cmake ../
make
```

3.3 Installing CPCd

Super-user permissions are required to install the daemon, cpclib, and the configuration file. These can be installed with the following commands:

```
make install
```

The following components will be installed:

- /usr/local/lib/libcpc.so.0.1
- /usr/local/lib/libcpc.so.1
- /usr/local/lib/libcpc.so
- /usr/local/include/sl_enum.h
- /usr/local/include/sl_cpc.h
- /usr/local/bin/cpcd
- /etc/cpcd.conf

Once installed, CPCd can be executed by invoking the `cpcd` command.

3.4 Configuring CPCd

When running the daemon without arguments, it starts with the default configuration file installed in the previous step. To specify a different configuration file, use the `--conf` argument. For example:

```
cpcd --conf <configuration file path>
```

3.5 Obtaining the Version of CPCd

If CPCd is started with the `-v` or `--version` argument, the daemon first prints the version of CPCd and exit. For example:

```
cpcd -version
```

3.6 Available Configurations

CPCd is configured in a key/value manner in the **cpcd.conf** file.

Table 1.1. CPCd configuration

Description	Key	Possible Values	Default Value	Mandatory
Instance Name	INSTANCE_NAME	string	cpcd_0	No
Bus type selection	BUS_TYPE	UART SPI	UART	Yes
SPI device file	SPI_DEVICE_FILE	Any path to SPI device file	/dev/spidev0.0	Yes if BUS_TYPE is SPI
SPI Chip Select GPIO #	SPI_CS_GPIO	Any GPIO # (1)	24	Yes if BUS_TYPE is SPI
SPI RX IRQ GPIO	SPI_RX_IRQ_GPIO	Any GPIO # (1)	23	Yes if BUS_TYPE is SPI
SPI Bitrate	SPI_DEVICE_BITRATE	Any (2)	1000000	Yes if BUS_TYPE is SPI
SPI Mode	SPI_DEVICE_MODE	SPI_MODE_0 (3) SPI_MODE_1 SPI_MODE_2 SPI_MODE_3	SPI_MODE_0	Yes if BUS_TYPE is SPI
UART Device File	UART_DEVICE_FILE	Any (4)	/dev/serial0	Yes if BUS_TYPE is UART
UART Baud Rate	UART_DEVICE_BAUD	1200 (5) 2400 4800 19200 38400 57600 115200	115200	Yes if BUS_TYPE is UART
UART Hardware Flow Control	UART_HARDFLOW	True or false	False	Yes if BUS_TYPE is UART
Trace to stdout	STDOUT_TRACE	True or false	False	No
Trace to a file located under TRACES_FOLDER	TRACE_TO_FILE	True or False	False	No
Destination folder when TRACE_TO_FILE is enabled	TRACES_FOLDER	Any path that the CPCd can access	./cpcd-traces	No
The maximum number of open file descriptors.	RLIMIT_NOFILE	Depends on the OS limit	2000	No
Disable the encryption over CPC endpoints	DISABLE_ENCRYPTION	True or false	False	No

(1) Make sure the CPC daemon has enough permissions to access this GPIO.

(2) This setting depends on various factors. The bitrate needs to satisfy both side requirements.

(3) Refer to section 2.6 for additional details.

(4) This setting depends on the Linux SOC.

(5) These baud rates are typical, but any value that meets both requirements can be used.

3.7 Available SPI Modes

The SPI_DEVICE_MODE configuration allows SPI clock polarity and phase to be configured, as shown in the following table.

Table 1.2. SPI Mode configuration

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)
SPI_MODE_0	0	0
SPI_MODE_1	0	1
SPI_MODE_2	1	0
SPI_MODE_3	1	1

4 Troubleshooting

If an error or a warning occurs during CPCd runtime, it prints to the console STDERR. If additional debugging is required, tracing can be enabled.

Note: Enabling traces may impact performance.

4.1 Tracing to the Standard Output (stdout)

When the configuration STDOUT_TRACE is enabled, the CPC daemon prints traces to the console.

4.2 Tracing to a File

When the configuration TRACE_TO_FILE is enabled. the CPC daemon prints traces to a file. The tracing file name contains the date and timestamp. This file is placed in the folder specified in the configuration TRACES_FOLDER.

The trace file has the following format:

```
trace-<year>-<month>-<day>-<hour>-<minute>-<second>.txt
```

The timestamp uses the operating system's local time zone.

Note: Only enable tracing to a file when debugging, as log file size increases over time.

5 Considerations

- The SPI driver uses a **sysfs** class GPIO as a chip select. Make sure the daemon has the proper permissions to access this GPIO.
- If the provided GPIO for the SPI chip select is already used by another driver, it needs to be deactivated and enabled as standard GPIO. In Linux this is usually done via the device tree.
- CPCd uses Unix sockets to exchange information with the Linux applications that use the CPC library. These sockets are stored under /etc/cpcd. Only users with the appropriate permissions should be able to access these sockets. CPCd inherits the permission of the user who starts the CPC daemon.
- Make sure no other application is using the serial bus at the same time as CPCd.
- Sensitive information can be exposed when tracing to a file is enable. Only enable tracing during development, for debugging purposes only. Refer to the TRACE_TO_FILE and STDOUT_TRACE configurations.

Smart. Connected. Energy-Friendly.

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com