cadence

Cadence I3C Master Controller Core Driver Quick Start Guide

Product Version 1.0.3

January 2024

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

1. Overview

This document describes the basics of how to get up and running with the Cadence IP Core Driver.

- Prepare the Cadence Platform Services (CPS)
- Prepare interrupt handling
- Build the Core Driver
- Use the Core Driver to initialize the Cadence IP
- Implement additional features using the Core Driver

2. Details

Here's how to get started:

1. *Prepare the Cadence Platform Services:*

   This is covered in detail in the Cadence Driver Porting Guide, which is available at doc/porting/porting_guide.pdf.

   CPS is a platform-specific code to connect a given Cadence driver to the rest of your system. Not all of the features of CPS are required for this driver. You only need to implement the functions for uncached access to memory:

   ```
   CPS_UncachedRead32, CPS_UncachedWrite32
   ```

   These functions will be used by the Core Driver to access the memory in a portable manner.

   The rest of the functions are not used by the Core Driver, thus don't need to be implemented.

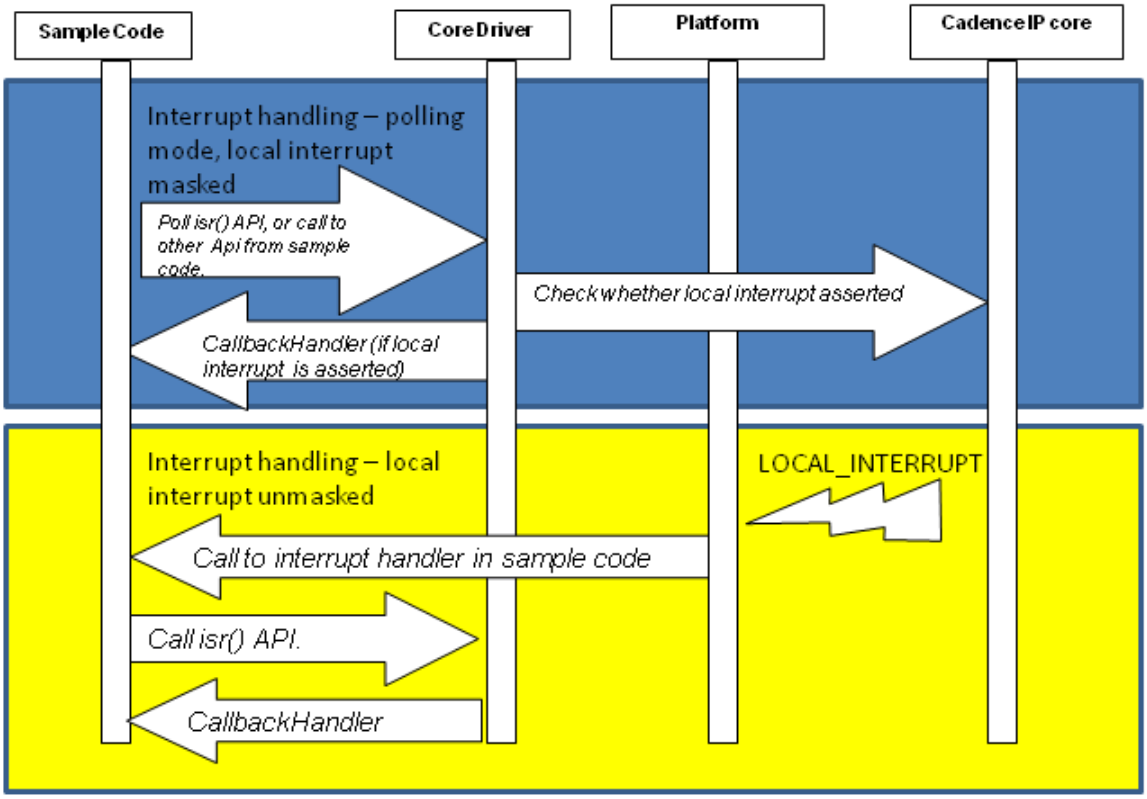
   After this step you should have an object file implementing the CPS for your target system which you can link with later.

2. *Prepare interrupt handling*

   The LOCAL_INTERRUPT interrupts are masked by default, and the status of these is polled as shown in the diagram below. The driver's *isr* API should be polled regularly to determine if there are any local interrupts active, and the driver will also check the status of local interrupts after other API calls. You can register a callback function when initializing the driver, this will be called if there are any local interrupts active.

If interrupts are unmasked (this can be done using *init*, or by calling *start* once the driver has been initialized), then an interrupt handler should be written and hooked up to the LOCAL_INTERRUPT output. This handler should then call the driver's interrupt handler: *isr*, which will in turn call any registered callback function. Details of how to hook up the interrupt handler will be processor and operating system specific, and are not covered in this documentation.

Figure 1. Local Interrupts



3. *Build the Core Driver*

   The driver and a sample makefile is provided in the CoreDriver directory. You will need to customise the makefile for your build environment, e.g. compiler, any necessary libraries etc.

   After this step you should have an object file i3c.o which can be linked with your own code.

4. *Use the Core Driver to initialize the Cadence IP*

Sample codes is available in the RefCode directory.

The sample code is designed to require only minor changes before it can be used. You will however need to change some default values in i3c_test.c.

```
/*
 * This should be the base address of the
 * Cadence I3C Master Controller Interface IP core on your system.
 */
#define I3C_REGS_BASE 0xFFFB0000

/*
 * This should be the IRQ number for the
 * Cadence I3C Master Controller Interface IP core on your system.
 */
#define I3C_IRQ_NUMBER 55
```

The sample code then calls *probe* to determine the size of memory which is required for private data. This memory must be provided by the code calling the driver, and may be allocated at compile time (e.g. by an array allocation) or dynamically. This private data will be passed to the driver as a parameter for each call to the API.

The sample code then prepares some example initialization parameters, e.g interrupt masks etc. You should replace these with any default settings you wish to configure, please see the user guide at doc/core_driver/i3c_driver_guide.pdf for more details on all available options.

The sample code then calls *init* to initialize the driver. Note that this does not start or restart the hardware, the hardware may be fully initialized by this stage using the default configuration.

As well as initializing the driver, the sample code also displays some diagnostic information using stdio library functions. If stdio is not available on your system, these calls should be stubbed out or replaced with an alternative (e.g. write to memory or local storage.)

After modifying the sample code to suit your environment, you should compile for your target system and link with the driver and CPS objects prepared previously, then download and run the code on your target system. This will be hardware and operating system specific, and is not covered in this documentation.