SMART ARM-based Microcontrollers

AT10942: SAM Configurable Custom Logic (CCL) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART SAM devices provides an interface for the configuration and management of the device's Configurable Custom Logic functionality.

The following peripheral is used by this module:

- CCL (Configurable Custom Logic)

The following devices can use this module:

- Atmel | SMART SAM L21
- Atmel | SMART SAM C20/C21

The outline of this documentation is as follows:

Table of Contents

1. Software License

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

This driver provides an interface for the Configurable Custom Logic functions on the device.

The Configurable Custom Logic (CCL) contains programmable logic which can be connected to the device pins, events, or internal peripherals.

Each LUT consists of three inputs, a truth table and optional synchronizer, filter and edge detector. Each LUT can generate an output as a user programmable logic expression with three inputs.

The output can be combinatorially generated from the inputs, or filtered to remove spike. An optional sequential module can be enabled. The inputs of sequential module are individually controlled by two independent, adjacent LUT(LUT0/LUT1, LUT2/LUT3, etc.) outputs, enabling complex waveform generation.

4. Special Considerations

There are no special considerations for this module.

5. Extra Information

For extra information, see Extra Information for CCL Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

For a list of examples related to this driver, see Examples for CCL Driver.

7. API Overview

7.1. Structure Definitions

7.1.1. Struct ccl_config

Configuration structure for CCL module.

Table 7-1 Members

Type	Name	Description
enum gclk_generator	clock_source	GCLK generator used to clock the peripheral
bool	run_in_standby	If `true`, the GCLK_CCL clock will not stop in standby sleep mode

7.1.2. Struct ccl_lut_config

Configuration structure for CCL LUT 0 to 3.

Table 7-2 Members

Type	Name	Description
bool	edge_selection_enable	If `true`, Edge detector is enabled
bool	event_input_enable	If `true`, LUT incoming event is enabled
bool	event_input_inverted_enable	If `true`, incoming event is inverted
bool	event_output_enable	If `true`, LUT event output is enabled
enum ccl_lut_filter_sel	filter_sel	Selection of the LUT output filter options
enum ccl_lut_input_src_sel	input0_src_sel	Selection of the input0 source
enum ccl_lut_input_src_sel	input1_src_sel	Selection of the input1 source
enum ccl_lut_input_src_sel	input2_src_sel	Selection of the input2 source
uint8_t	truth_table_value	The value of truth logic as a function of inputs IN[2:0]

7.2. Function Definitions

7.2.1. Initialize and Reset CCL Module

7.2.1.1. Function ccl_init()

Initializes CCL module.

```
void ccl_init(
        struct ccl_config *const config)
```

Resets all registers in the MODULE to their initial state, and then enable the module.

7.2.1.2. Function ccl_get_config_defaults()

Initializes all members of a CCL configuration structure to safe defaults.

```
void ccl_get_config_defaults(
        struct ccl_config *const config)
```

Initializes all members of a given Configurable Custom Logic configuration structure to safe and known default values. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- GCLK_CLL will be stopped in standby sleep mode
- Generator 0 is the default GCLK generator

Table 7-3 Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to initialize to default values

7.2.1.3. Function ccl_module_reset()

Resets CCL module.

```
void ccl_module_reset( void )
```

Resets all registers in the MODULE to their initial state, and the CCL will be disabled.

7.2.2. Enable and Disable CCL Module

7.2.2.1. Function ccl_module_enable()

Enables CCL module.

```
void ccl_module_enable( void )
```

Enable the peripheral.

7.2.2.2. Function ccl_module_disable()

Disables CCL module.

```
void ccl_module_disable( void )
```

Disables the peripheral.

7.2.3. Configure LUT

7.2.3.1. Function ccl_seq_config()

Writes sequential selection to the hardware module.

```
enum status_code ccl_seq_config(
        const enum ccl_seq_id number,
        const enum ccl_seq_selection seq_selection)
```

Writes a given sequential selection configuration to the hardware module.

Note: This function can only be used when the CCL module is disabled.

Table 7-4 Parameters

Data direction	Parameter name	Description
[in]	seq_selection	Enum for the sequential selection configuration
[in]	number	SEQ unit number to config

7.2.3.2. Function ccl_lut_get_config_defaults()

Initializes all members of LUT configuration structure to safe defaults.

```
void ccl_lut_get_config_defaults(
        struct ccl_lut_config *const config)
```

Initializes all members of LUT configuration structure to safe defaults. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- Truth table value is 0x00
- LUT event output is disabled
- LUT incoming event is disabled
- LUT incoming event is not inverted
- The input IN[2:0] source is masked
- The edge detector is disabled
- The LUT output filter is disabled

Table 7-5 Parameters

Data direction	Parameter name	Description
[out]	config	LUT configuration structure to initialize to default values

7.2.3.3. Function ccl_lut_set_config()

Writes LUT configuration to the hardware module.

```
enum status_code ccl_lut_set_config(
        const enum ccl_lut_id number,
        struct ccl_lut_config *const config)
```

Writes a given LUT configuration to the hardware module.

Note: This function can only be used when the CCL module is disabled.

Table 7-6 Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to the LUT configuration struct
[in]	number	LUT number to config

7.2.4. Enable and Disable LUT

7.2.4.1. Function ccl_lut_enable()

Enables an LUT that was previously configured.

```
void ccl_lut_enable(
        const enum ccl_lut_id number)
```

Enables an LUT that was previously configured via a call to ccl_lut_set_config function.

Table 7-7 Parameters

Data direction	Parameter name	Description
[in]	number	LUT number to enable

7.2.4.2. Function ccl_lut_disable()

Disables an LUT that was previously enabled.

```
void ccl_lut_disable(
        const enum ccl_lut_id number)
```

Disables an LUT that was previously enabled via a call to ccl_lut_enable().

Table 7-8 Parameters

Data direction	Parameter name	Description
[in]	number	LUT number to enable

7.3. Enumeration Definitions

7.3.1. Enum ccl_lut_filter_sel

Enum for the LUT output filter options.

Table 7-9 Members

Enum value	Description
CCL_LUT_FILTER_DISABLE	Filter disabled
CCL_LUT_FILTER_SYNC	Synchronizer enabled
CCL_LUT_FILTER_ENABLE	Filter enabled

7.3.2. Enum ccl_lut_id

Table 7-10 Members

Enum value	Description
CCL_LUT_0	CCL LUT 0
CCL_LUT_1	CCL LUT 1

Enum value	Description
CCL_LUT_2	CCL LUT 2
CCL_LUT_3	CCL LUT 3

7.3.3. Enum ccl_lut_input_src_sel

Enum for the LUT Input source selection.

Table 7-11 Members

Enum value	Description
CCL_LUT_INPUT_SRC_MASK	Masked input
CCL_LUT_INPUT_SRC_FEEDBACK	Feedback input source
CCL_LUT_INPUT_SRC_LINK	Linked LUT input source
CCL_LUT_INPUT_SRC_EVENT	Event input source
CCL_LUT_INPUT_SRC_IO	I/O pin input source
CCL_LUT_INPUT_SRC_AC	AC input source
CCL_LUT_INPUT_SRC_TC	TC input source
CCL_LUT_INPUT_SRC_ALTTC	Alternative TC input source
CCL_LUT_INPUT_SRC_TCC	TCC input source
CCL_LUT_INPUT_SRC_SERCOM	SERCOM input source

7.3.4. Enum ccl_seq_id

Table 7-12 Members

Enum value	Description
CCL_SEQ_0	CCL SEQ 0
CCL_SEQ_1	CCL SEQ 1

7.3.5. Enum ccl_seq_selection

Enum for the sequential selection configuration.

Table 7-13 Members

Enum value	Description
CCL_SEQ_DISABLED	Sequential logic is disabled
CCL_SEQ_D_FLIP_FLOP	D flip flop
CCL_SEQ_JK_FLIP_FLOP	JK flip flop

Enum value	Description
CCL_SEQ_D_LATCH	D latch
CCL_SEQ_RS_LATCH	RS latch

8. Extra Information for CCL Driver

8.1. Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Description
CCL	Configurable Custom Logic

8.2. Dependencies

This driver has no dependencies.

8.3. Errata

There are no errata related to this driver.

8.4. Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial Release

9. Examples for CCL Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM Configurable Custom Logic (CCL) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for CCL - Basic

9.1. Quick Start Guide for CCL - Basic

In this use case, the LUT0 and LUT1 input source is configured as I/O pin. The LUT0 and LUT1 pair is connected to internal sequential logic, which is configured as D flip flop mode.

9.1.1. Setup

9.1.1.1. Prerequisites

There are no special setup requirements for this use-case.

9.1.1.2. Code

Copy-paste the following setup code to your user application:

```
void configure_ccl(void)
{
    struct ccl_config conf;

    ccl_get_config_defaults(&conf);

    ccl_init(&conf);
}

void configure_ccl_lut0(void)
{
    struct ccl_lut_config conf;

    ccl_lut_get_config_defaults(&conf);

    conf.truth_table_value = 0x02;
    conf.input0_src_sel = CCL_LUT_INPUT_SRC_IO;
    conf.input1_src_sel = CCL_LUT_INPUT_SRC_IO;
    conf.input2_src_sel = CCL_LUT_INPUT_SRC_IO;
    conf.filter_sel = CCL_LUTCTRL_FILTSEL_FILTER;

    struct system_pinmux_config lut0_input_pin0_conf,
lut0_input_pin1_conf, lut0_input_pin2_conf;
    system_pinmux_get_config_defaults(&lut0_input_pin0_conf);
    system_pinmux_get_config_defaults(&lut0_input_pin1_conf);
    system_pinmux_get_config_defaults(&lut0_input_pin2_conf);
    lut0_input_pin0_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    lut0_input_pin0_conf.mux_position = CCL_LUT0_IN0_MUX;
    lut0_input_pin1_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    lut0_input_pin1_conf.mux_position = CCL_LUT0_IN1_MUX;
    lut0_input_pin2_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    lut0_input_pin2_conf.mux_position = CCL_LUT0_IN2_MUX;
    system_pinmux_pin_set_config(CCL_LUT0_IN0_PIN, &lut0_input_pin0_conf);
    system_pinmux_pin_set_config(CCL_LUT0_IN1_PIN, &lut0_input_pin1_conf);
```

```
    system_pinmux_pin_set_config(CCL_LUT0_IN2_PIN, &lut0_input_pin2_conf);
    struct system_pinmux_config lut0_out_pin_conf;
    system_pinmux_get_config_defaults(&lut0_out_pin_conf);
    lut0_out_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_OUTPUT;
    lut0_out_pin_conf.mux_position = CCL_LUT0_OUT_MUX;
    system_pinmux_pin_set_config(CCL_LUT0_OUT_PIN, &lut0_out_pin_conf);

    ccl_lut_set_config(CCL_LUT_0, &conf);
}

void configure_ccl_lut1(void)
{
    struct ccl_lut_config conf;

    ccl_lut_get_config_defaults(&conf);

    conf.truth_table_value = 0x02;
    conf.input0_src_sel = CCL_LUT_INPUT_SRC_IO;
    conf.input1_src_sel = CCL_LUT_INPUT_SRC_IO;
    conf.input2_src_sel = CCL_LUT_INPUT_SRC_IO;
    conf.filter_sel = CCL_LUTCTRL_FILTSEL_FILTER;

    struct system_pinmux_config lut1_input_pin0_conf,
lut1_input_pin1_conf, lut1_input_pin2_conf;
    system_pinmux_get_config_defaults(&lut1_input_pin0_conf);
    system_pinmux_get_config_defaults(&lut1_input_pin1_conf);
    system_pinmux_get_config_defaults(&lut1_input_pin2_conf);
    lut1_input_pin0_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    lut1_input_pin0_conf.mux_position = CCL_LUT1_IN0_MUX;
    lut1_input_pin1_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    lut1_input_pin1_conf.mux_position = CCL_LUT1_IN1_MUX;
    lut1_input_pin2_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    lut1_input_pin2_conf.mux_position = CCL_LUT1_IN2_MUX;
    system_pinmux_pin_set_config(CCL_LUT1_IN0_PIN, &lut1_input_pin0_conf);
    system_pinmux_pin_set_config(CCL_LUT1_IN1_PIN, &lut1_input_pin1_conf);
    system_pinmux_pin_set_config(CCL_LUT1_IN2_MUX, &lut1_input_pin2_conf);
    struct system_pinmux_config lut1_out_pin_conf;
    system_pinmux_get_config_defaults(&lut1_out_pin_conf);
    lut1_out_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_OUTPUT;
    lut1_out_pin_conf.mux_position = CCL_LUT1_OUT_MUX;
    system_pinmux_pin_set_config(CCL_LUT1_OUT_PIN, &lut1_out_pin_conf);

    ccl_lut_set_config(CCL_LUT_1, &conf);
}
```

Add to user application initialization (typically the start of `main()`):

```
configure_ccl();
configure_ccl_lut0();
configure_ccl_lut1();
ccl_seq_config(CCL_SEQ_0, CCL_SEQ_D_FLIP_FLOP);
```

```
ccl_lut_enable(CCL_LUT_0);
ccl_lut_enable(CCL_LUT_1);
ccl_module_enable();
```

9.1.1.3. Workflow

1. Creates a CCL configuration struct, which can be filled out to adjust the configuration of CCL.

```
struct ccl_config conf;
```

2. Settings and fill the CCL configuration struct with the default settings.

```
ccl_get_config_defaults(&conf);
```

3. Initializes CCL module.

```
ccl_init(&conf);
```

4. Creates a LUT configuration struct, which can be filled out to adjust the configuration of LUT0.

```
struct ccl_lut_config conf;
```

   1. Fill the LUT0 configuration struct with the default settings.

```
ccl_lut_get_config_defaults(&conf);
```

5. Adjust the LUT0 configuration struct.

```
conf.truth_table_value = 0x02;
conf.input0_src_sel = CCL_LUT_INPUT_SRC_IO;
conf.input1_src_sel = CCL_LUT_INPUT_SRC_IO;
conf.input2_src_sel = CCL_LUT_INPUT_SRC_IO;
conf.filter_sel = CCL_LUTCTRL_FILTSEL_FILTER;
```

6. Set up LUT0 input and output pin.

```
struct system_pinmux_config lut0_input_pin0_conf,
lut0_input_pin1_conf, lut0_input_pin2_conf;
system_pinmux_get_config_defaults(&lut0_input_pin0_conf);
system_pinmux_get_config_defaults(&lut0_input_pin1_conf);
system_pinmux_get_config_defaults(&lut0_input_pin2_conf);
lut0_input_pin0_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
lut0_input_pin0_conf.mux_position = CCL_LUT0_IN0_MUX;
lut0_input_pin1_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
lut0_input_pin1_conf.mux_position = CCL_LUT0_IN1_MUX;
lut0_input_pin2_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
lut0_input_pin2_conf.mux_position = CCL_LUT0_IN2_MUX;
system_pinmux_pin_set_config(CCL_LUT0_IN0_PIN, &lut0_input_pin0_conf);
system_pinmux_pin_set_config(CCL_LUT0_IN1_PIN, &lut0_input_pin1_conf);
system_pinmux_pin_set_config(CCL_LUT0_IN2_PIN, &lut0_input_pin2_conf);
struct system_pinmux_config lut0_out_pin_conf;
system_pinmux_get_config_defaults(&lut0_out_pin_conf);
lut0_out_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_OUTPUT;
lut0_out_pin_conf.mux_position = CCL_LUT0_OUT_MUX;
system_pinmux_pin_set_config(CCL_LUT0_OUT_PIN, &lut0_out_pin_conf);
```

7. Writes LUT0 configuration to the hardware module.

```
ccl_lut_set_config(CCL_LUT_0, &conf);
```

8. Creates a LUT configuration struct, which can be filled out to adjust the configuration of LUT1.

```
struct ccl_lut_config conf;
```

   1. Fill the LUT1 configuration struct with the default settings.

```
ccl_lut_get_config_defaults(&conf);
```

9. Adjust the LUT1 configuration struct.

```
conf.truth_table_value = 0x02;
conf.input0_src_sel = CCL_LUT_INPUT_SRC_IO;
conf.input1_src_sel = CCL_LUT_INPUT_SRC_IO;
conf.input2_src_sel = CCL_LUT_INPUT_SRC_IO;
conf.filter_sel = CCL_LUTCTRL_FILTSEL_FILTER;
```

10. Set up LUT1 input and output pin.

```
struct system_pinmux_config lut1_input_pin0_conf,
lut1_input_pin1_conf, lut1_input_pin2_conf;
system_pinmux_get_config_defaults(&lut1_input_pin0_conf);
system_pinmux_get_config_defaults(&lut1_input_pin1_conf);
system_pinmux_get_config_defaults(&lut1_input_pin2_conf);
lut1_input_pin0_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
lut1_input_pin0_conf.mux_position = CCL_LUT1_IN0_MUX;
lut1_input_pin1_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
lut1_input_pin1_conf.mux_position = CCL_LUT1_IN1_MUX;
lut1_input_pin2_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
lut1_input_pin2_conf.mux_position = CCL_LUT1_IN2_MUX;
system_pinmux_pin_set_config(CCL_LUT1_IN0_PIN, &lut1_input_pin0_conf);
system_pinmux_pin_set_config(CCL_LUT1_IN1_PIN, &lut1_input_pin1_conf);
system_pinmux_pin_set_config(CCL_LUT1_IN2_MUX, &lut1_input_pin2_conf);
struct system_pinmux_config lut1_out_pin_conf;
system_pinmux_get_config_defaults(&lut1_out_pin_conf);
lut1_out_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_OUTPUT;
lut1_out_pin_conf.mux_position = CCL_LUT1_OUT_MUX;
system_pinmux_pin_set_config(CCL_LUT1_OUT_PIN, &lut1_out_pin_conf);
```

11. Writes LUT1 configuration to the hardware module.

```
ccl_lut_set_config(CCL_LUT_1, &conf);
```

12. Configure the sequential logic with the D flip flop mode.

```
ccl_seq_config(CCL_SEQ_0, CCL_SEQ_D_FLIP_FLOP);
```

9.1.2. Use Case

9.1.2.1. Code

Copy-paste the following code to your user application:

```
while (true) {
    /* Do nothing */
}
```

10. Document Revision History

Doc. Rev.	Date	Comments
42448A	06/2015	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**