APPLICATION NOTE

AT05567: TC Capture on External Interrupt with the Event System on SAM D20

ASF PROJECT DOCUMENTATION

Preface

This application note shows how to use the event system to capture an external interrupt signal with the Timer/Counter in SAM D20. Specifically, we will build and example where we measure the length of a button push, and then light a LED for the same period of time.

Features:

- Setting up the External Interrupt Controller for a mechanical button push interrupt
- Routing a signal from one peripheral to another with the Event System
- Configuring the Timer/Counter to do pulse-width captures
- Outputting the result on a LED by using Timer/Counter compare

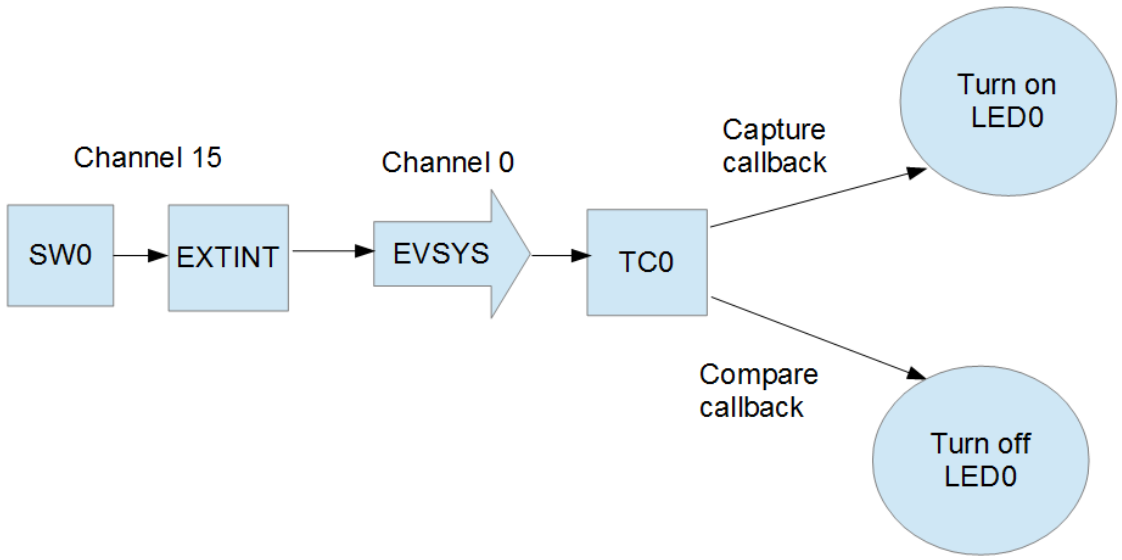
Table of Contents

1. Introduction

This application note demonstrates how to use the device's Timer/Counter modules to do a pulse-width capture, without involving the CPU. The Event System is configured to route an external interrupt to the Timer/Counter, which will then capture the time interval in which a button is pressed. When the button is released, a LED will light up in the same amount of time as the button was pushed, by utilizing the compare functionality of the Timer/Counter.

Figure 1-1: Block Diagram of the Application on page 3 illustrates how the different modules will interact in the example described in this application note.

Figure 1-1. Block Diagram of the Application



1.1 Prerequisites

Running the example described in this application note requires:

- Atmel® Studio 6.1 Service Pack 2
- Atmel Software Framework 3.12.1 or higher
- SAM D20 Xplained PRO Evaluation kit with USB cable

2. Example Overview

2.1 Functionality

The SAM D20 Timer/Counter module can capture a signal entirely without CPU intervention. This makes it possible to do very accurate captures with low latency.

This is demonstrated in the following example. The External Interrupt module is configured to generate an event when button SW0 is pushed. SW0 is connected to PA15, which can generate interrupts on EXTINT[15]. This is routed to Timer/Counter TC0 by the Event System channel 15. Then the capture is done automatically by having TC0 in PWP (Pulse-Width Period) mode. Capture Channel 0 (CC0) captures the pulse-width (and Capture Channel 1 (CC1) can capture the period if CC1 is enabled). Afterwards, capture on CC0 is disabled, so that the channel can be used to give a compare match interrupt when the button push interval has passed.

2.2 Peripherals

The following peripherals are used by this example:

- Timer/Counter (TC)[1]
- External Interrupt (EXTINT)[2]
- Event System (EVSYS)[3]

2.3 Configurations

2.3.1 Configure the 16-bit TC0 Module for Capture

1. Declare the following structs globally:

```
/* TC module struct */
struct tc_module tc0_module;

/* TC config struct */
struct tc_config tc0_config;
```

Create a function `tc_configuration()` where you configure Timer/Counter0 as follows:

- Prescaler: 1024
- Enable capture on channel 0
- Inverted event input, as the signal from the button is active low
- Event action PWP (Pulse-Width and Period)

2. This is done by using the structs you just declared like this:

```
/* Configure 16-bit TC0 module for capture */
void tc_configuration(void)
{
    tc_reset(&tc0_module);
    tc_get_config_defaults(&tc0_config);
    tc0_config.clock_prescaler              = TC_CLOCK_PRESCALER_DIV1024;
    tc0_config.enable_capture_on_channel[0] = true;
    tc0_config.invert_event_input           = true;
    tc0_config.event_action                 = TC_EVENT_ACTION_PWP;
```

[1] http://www.atmel.com/Images/Atmel-42123-SAM-D20-Timer-Counter-Driver-TC_Application-Note_AT03263.pdf
[2] http://www.atmel.com/Images/Atmel-42112-SAM-D20-External-Interrupt-Driver-EXTINT_Application-Note_AT03246.pdf
[3] http://www.atmel.com/Images/Atmel-42108-SAM-D20-Event-System-Driver_Application-Note_AT03245.pdf

```
    tc_init(&tc0_module, TC0, &tc0_config);
```

3. Generate events on captures:

```
struct tc_events tc_event = { .generate_event_on_compare_channel[0] = true,
     .generate_event_on_compare_channel[1] = true,
     .on_event_perform_action = true };
tc_enable_events(&tc0_module, &tc_event);
```

4. Set the capture callback function:

```
/* Callback function for capture */
tc_register_callback(&tc0_module, tc_capture_callback_function,
    TC_CALLBACK_CC_CHANNEL0);
tc_enable_callback(&tc0_module, TC_CALLBACK_CC_CHANNEL0);
```

Note With these clock settings the timer will overflow after about 8 seconds. The Overflow Interrupt flag (INTFLAG.OVF) can be checked to see if a count overflow has occurred.

5. The resulting `tc_configuration()` function should be something like this:

```
/* Configure 16-bit TC0 module for capture */
void tc_configuration(void)
{
    tc_reset(&tc0_module);
    tc_get_config_defaults(&tc0_config);
    tc0_config.clock_prescaler              = TC_CLOCK_PRESCALER_DIV1024;
    tc0_config.enable_capture_on_channel[0] = true;
    tc0_config.invert_event_input           = true;
    tc0_config.event_action                 = TC_EVENT_ACTION_PWP;
    tc_init(&tc0_module, TC0, &tc0_config);

    struct tc_events tc_event = { .generate_event_on_compare_channel[0] = true,
         .generate_event_on_compare_channel[1] = true,
         .on_event_perform_action = true };
    tc_enable_events(&tc0_module, &tc_event);

    /* Callback function for capture */
    tc_register_callback(&tc0_module, tc_capture_callback_function,
        TC_CALLBACK_CC_CHANNEL0);
    tc_enable_callback(&tc0_module, TC_CALLBACK_CC_CHANNEL0);
}
```

2.3.2 Configure the External Interrupt Controller

1. Create a function for configuring external interrupts, and declare a configuration struct:

```
/* Configure external interrupt controller */
void extint_configuration(void)
{
    struct extint_chan_conf extint_chan_config;
```

The External Interrupt Controller should be configured with the following settings:

- Interrupt from GPIO pin PA15 (EXTINT 15)

- Set MUX to GPIO pin PA15
- Enable pull-up (default)
- Detect high, to be able to notice when the signal goes high (button released)

2. This is done by using the configuration struct you just declared:

```
extint_chan_get_config_defaults(&extint_chan_config);
extint_chan_config.gpio_pin            = PIN_PA15A_EIC_EXTINT15;
extint_chan_config.gpio_pin_mux        = MUX_PA15A_EIC_EXTINT15;
extint_chan_config.detection_criteria  = EXTINT_DETECT_HIGH;
extint_chan_set_config(15, &extint_chan_config);
extint_enable();
```

3. Generate event on detect in channel 15 by declaring another configuration struct which is used like this:

```
/* Configure external interrupt module to be event generator */
struct extint_events extint_event_conf;
extint_event_conf.generate_event_on_detect[15] = true;
extint_enable_events(&extint_event_conf);
```

4. The resulting `extint_configuration()` function should be something like this:

```
/* Configure external interrupt controller */
void extint_configuration(void)
{
    struct extint_chan_conf extint_chan_config;

    extint_chan_get_config_defaults(&extint_chan_config);
    extint_chan_config.gpio_pin            = PIN_PA15A_EIC_EXTINT15;
    extint_chan_config.gpio_pin_mux        = MUX_PA15A_EIC_EXTINT15;
    extint_chan_config.detection_criteria  = EXTINT_DETECT_HIGH;
    extint_chan_set_config(15, &extint_chan_config);
    extint_enable();

    /* Configure external interrupt module to be event generator */
    struct extint_events extint_event_conf;
    extint_event_conf.generate_event_on_detect[15] = true;
    extint_enable_events(&extint_event_conf);
}
```

2.3.3 Configure the Event System

1. Create a function for configuring the event system and initialize the event system:

```
/* Configure event system: */
void evsys_configuration(void)
{
    events_init();
```

2. Create a configuration struct for configuring the event system user:

```
struct events_user_config event_user_conf;
```

The Event System User should be configured with these settings:

- Event system channel 0 (default)

- Event system user: TC0

3. Use the configuration struct you just declared to do it:

```
events_user_get_config_defaults(&event_user_conf);
event_user_conf.event_channel_id = EVENT_CHANNEL_0;
events_user_set_config(EVSYS_ID_USER_TC0_EVU, &event_user_conf);
```

4. Create a configuration struct for configuring the event system channel:

```
struct events_chan_config events_ch0_conf;
```

The Event System Channel should be configured with these settings:

- Event generator ID: EXTINT 15
- No edge detection, as the whole signal is routed directly to the TC anyway (default)
- Asynchronous (default)

5. Use the configuration struct you just declared like this:

```
events_chan_get_config_defaults(&events_ch0_conf);
events_ch0_conf.generator_id   = EVSYS_ID_GEN_EIC_EXTINT_15; //SW0
events_chan_set_config(EVENT_CHANNEL_0, &events_ch0_conf);
```

6. The resulting `evsys_configuration()` function should be something like this:

```
/* Configure event system: */
void evsys_configuration(void)
{
    events_init();
    /* Configure event system user. Event channel 0 */
    struct events_user_config event_user_conf;

    events_user_get_config_defaults(&event_user_conf);
    event_user_conf.event_channel_id = EVENT_CHANNEL_0;
    events_user_set_config(EVSYS_ID_USER_TC0_EVU, &event_user_conf);

    /* Setup PA15 (SW0) as input to event system channel 0, asynchronous */
    struct events_chan_config events_ch0_conf;

    events_chan_get_config_defaults(&events_ch0_conf);
    events_ch0_conf.generator_id   = EVSYS_ID_GEN_EIC_EXTINT_15; //SW0
    events_chan_set_config(EVENT_CHANNEL_0, &events_ch0_conf);
}
```

2.4 Callback Functions

Two different callback functions are used in this example. Initially, the callback function of the Timer/Counter is set to be the `tc_capture_callback_function()`. This is entered when the button has been released, and the pulse-width is ready to be processed. The next interrupt will happen once the pulse-width time interval has passed, and this time the `tc_compare_callback_function()` should be entered.

2.4.1 Capture Callback Function

This callback function is entered when the button is released. The following must be done:

- Store pulse width (captured into CC0) in variable `button_time`
- Disable capture on channel 0 to enable compare functionality (will also reset the timer)
- Set compare value (CC0) to `button_time`
- Register the `tc_compare_callback_function()` as callback for TC0
- Turn on LED0

The callback function can be implemented as below:

```
/* Capture callback function */
void tc_capture_callback_function(struct tc_module *const module_inst)
{
    uint32_t button_time = tc_get_capture_value(&tc0_module,
        TC_COMPARE_CAPTURE_CHANNEL_0);

    /* Configure timer to wait for compare match */
    tc_disable(&tc0_module); //Disable TC0 module to be able to configure it
    tc0_config.enable_capture_on_channel[0] = false;
    tc_init(&tc0_module, TC0, &tc0_config);
    /*Set the value of CC0 as it has been reset*/
    tc_set_compare_value(&tc0_module, TC_COMPARE_CAPTURE_CHANNEL_0, button_time);

    /* Change callback function to tc_led_callback_function */
    tc_register_callback(&tc0_module, tc_compare_callback_function,
        TC_CALLBACK_CC_CHANNEL0);
    tc_enable_callback(&tc0_module, TC_CALLBACK_CC_CHANNEL0);

    tc_enable(&tc0_module); //Enable TC0 module again

    /* LED0 turned on for as long as the button was pushed (button_time) */
    port_pin_set_output_level(LED_0_PIN, LED_0_ACTIVE);
}
```

2.4.2 Compare Callback Function

This callback function will be entered when the timer's COUNT register matches the CC0 register (`button_time`) and capture mode is off. The following must be done:

- Turn off LED0
- Enable capture on channel 0 (will also reset the timer)
- Register `tc_capture_callback_function()` as callback for TC0
- Generate events on captures

The callback function can be implemented as below:

```
/* Compare callback function */
void tc_compare_callback_function(struct tc_module *const module_inst)
{
    /* LED0 turned off */
    port_pin_set_output_level(LED_0_PIN, LED_0_INACTIVE);

    /* Configure timer back to pulse-width capture operation by setting the timer
    to wait for compare match*/
    tc_disable(&tc0_module);
    tc0_config.enable_capture_on_channel[0] = true;
```

```
    tc_init(&tc0_module, TC0, &tc0_config);

    /*Change callback function back to tc_button_callback_function*/
    tc_register_callback(&tc0_module, tc_capture_callback_function,
        TC_CALLBACK_CC_CHANNEL0);
    tc_enable_callback(&tc0_module, TC_CALLBACK_CC_CHANNEL0);

    struct tc_events tc_event = { .generate_event_on_compare_channel[0] = true,
        .generate_event_on_compare_channel[1] = true,
        .on_event_perform_action = true };
    tc_enable_events(&tc0_module, &tc_event);
    tc_enable(&tc0_module); //Enable TC0 module again
}
```

2.5 Main Routine

After configuring the different modules as described, the only part that remains is to enable the TC0 module, and enable interrupts:

```
/* Enable TC0 module */
tc_enable(&tc0_module);

/*Enable interrupts*/
system_interrupt_enable(TC0_IRQn);
system_interrupt_enable_global();
```

All together, with all the function calls and an empty `while`, your `main()` should look like this:

```
int main (void)
{
    /*Initialization and configuration*/
    system_init();
    tc_configuration();
    extint_configuration();
    evsys_configuration();

    /* Enable TC0 module */
    tc_enable(&tc0_module);

    /*Enable interrupts*/
    system_interrupt_enable(TC0_IRQn);
    system_interrupt_enable_global();

    while (1) {
        /*Wait for button to get pushed*/
    }
}
```

2.6 Summary

The event system of the SAM D20 makes it possible to route a signal from one peripheral to another without involving the CPU. In this application note we have seen how this can be used to do a capture of an external signal.

When the button is pushed, the event system passes the signal directly from the External Interrupt Controller to the Timer/Counter. This makes it possible to achieve a very low latency on captures. Additionally, power consumption can be reduced as the CPU can be left sleeping even while the capture is done. This is however not implemented in this example.

3. License

4. Revision History

Doc. Rev.	Date	Comments
A	05/2014	Initial release

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