UNIVERSITÄT AUGSBURG

A Simple Capture/Compare Timer

Florian Kluge

Report 2015-01 Juni 2015

INSTITUT FÜR INFORMATIK
D-86135 AUGSBURG

SCCT is a Simple Capture/Compare Timer written in Verilog. It provides multiple capture/compare channels that use a common counter. Events occurring in the single channels thus can be related to a global time base. SCCT is developed as an IP core that can be attached to the Altera Avalon bus.

Contents

1 Overview

Parts of our research are dealing with the development of real-time capable multicore processors for embedded systems. Prototypes of these processors are deployed on field-programmable gate arrays (FPGAs). Use cases for such systems, like management of an internal combustion engine, require that certain reactions performed by the computer are well aligned in time with input events. On microcontroller level, this can be achieved by utilising capture/compare timers. Neither the Altera IP core suite (which we are using in our research) provides one, nor did we find a free one available on the Internet. We provide the SCCT IP core to the public in the hope that someone might find it useful for his own work.

The source code of SCCT can be downloaded from OpenCores[1]. It is made available under the conditions of the GNU General Public Licence Version 3[2].

The remainder of this report is structured as follows: In the next chapter, we present the structure of SCCT and how the single modules interact. Chapter 3 describes the programming interface of SCCT. Finally, in chapter 4, we describe additional tools that are part of the SCCT package.

[1] http://opencores.org/project,scct
[2] http://www.gnu.org/copyleft/gpl.html

2 Structure and Functionality

2.1 Overview

SCCT comprises one counter and multiple channel modules. A coarse overview of the whole SCCT module is shown in figure 2.1. The counter module provides the current counter value to all channels. Each channel is connected to an input and an output pin.

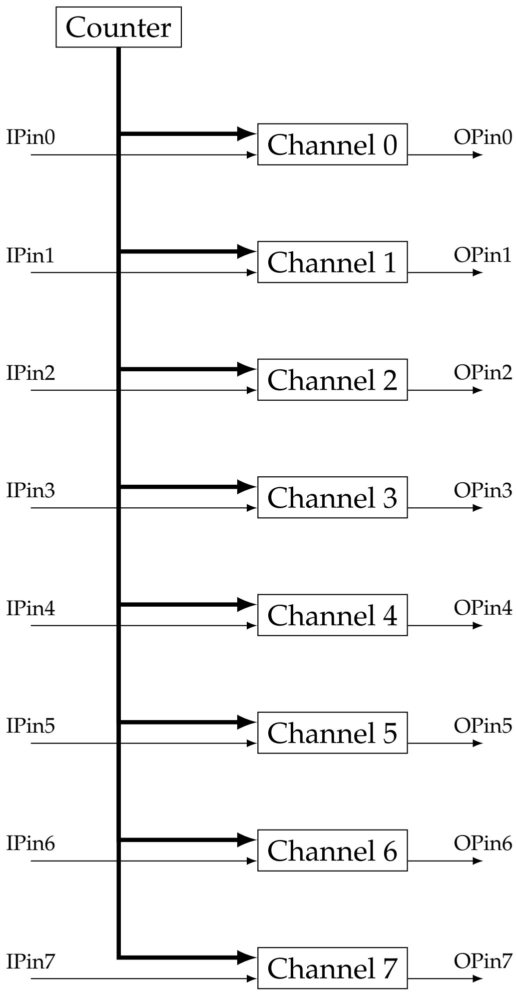

Figure 2.1: Overview of the simple capture/compare timer

2.2 Counter Unit

The counter unit provides the common counter for all channels of SCCT. Its internal structure is depicted in figure 2.2. The `counter` register is the heart of the counter unit. Advance of the `counter` register is controlled by a prescaler. Setting of the prescaler is performed via the `prescaler` register. Prescaler comparisons are based on the prescaler shadow register `prescaler_sh`. The `prescaler count` register is incremented each clock cycle. If it reaches the value of the `prescaler_sh` register, an event is generated that has several consequences: (1) The counter register is incremented by 1, (2) a `counter_changed` output signal is set for one cycle to notify connected units about this event, and (3) the current value of the `prescaler` register is copied to the `prescaler_sh` register. Increment events for the `counter` register thus are generated each `prescaler` + 1 clock cycles. Changing the `prescaler` register will not immediately affect the counter, but just when the current prescale interval elapses.

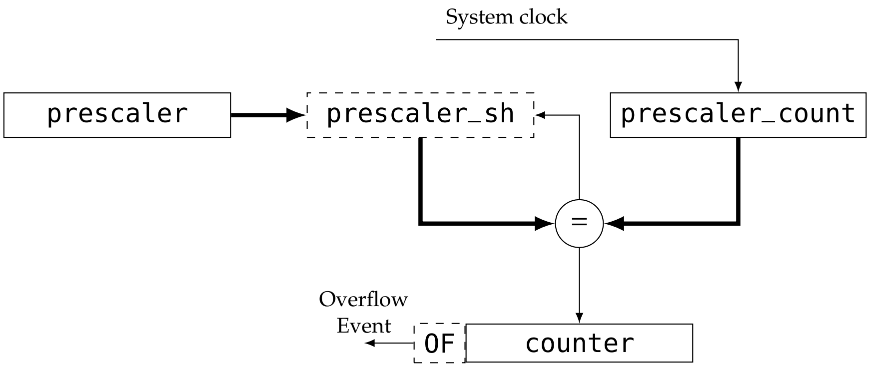

Figure 2.2: The counter unit

If the counter overflows, an interrupt can be raised. At the moment, an overflow can only occur if the counter value exceeds the maximum value that can be represented by the width of the counter register. Overflow detection is implemented in the following manner: The actual width of the counter register is one bit more than required by the counter resolution. This higher-most bit (OF) is automatically set to 1 when an overflow occurs (concerning the counter's resolution) and directly triggers the overflow event. It is reset to 0 after the overflow event has been registered. All other bits of the counter register are automatically set to 0 due to the overflow.

2.3 Capture/Compare Channel

Each capture/compare channel of SCCT is constructed in the same manner. The structure of a single channel is depicted in figure 2.3. The channel receives the `counter` value and the `counter_changed` signal that are generated by the counter unit. A channel can be configured either as input capture or output compare.

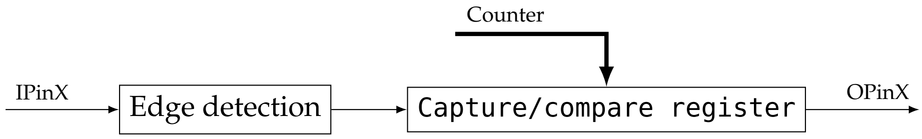

Figure 2.3: Structure of a single capture/compare channel

2.3.1 Input Capture

If the channel is configured as input capture channel, it reacts on level changes of its input pin. It can react on rising, falling, or both types of edges. Edge detection is performed each clock cycle by comparing the level of the current cycle with the one of the previous cycle that is stored in a register. If an edge of interest is detected, an input capture event is emitted. The current counter value then is stored in the `capture/compare` register. Additionally, an interrupt can be raised.

2.3.2 Output Compare

In output compare mode, the channel reacts if the counter value reaches the value that was written to the *capture/compare* register. The reaction is only triggered in that cycle, during which the counter value changed, indicated by the `counter_changed` signal. On a reaction, it can either toggle the level of the output pin, or set it to low or high state. Like in input capture mode, an interrupt can be raised.

The `counter_changed` signal as an additional trigger condition was added to deal with the following scenario: Assume that no `counter_changed` signal is used, and that the `prescaler` is set to a very large value. If the counter reaches the value that is set in the `capture/compare` register of an OC channel, an interrupt is raised, and a software interrupt handler is executed. At some point, the handler resets the interrupt status of the channel and returns. If the `prescaler` value is larger than the execution time of the interrupt handler (in clock cycles), another interrupt would be raised immediately as the `counter` value still equals the `capture/compare` value of the channel. This behaviour is prohibited through the use of a `counter_changed` signal.

2.4 Top-level Module

The top-level module has two tasks: On the one hand, it connects the channels with the outputs of the counter module. On the other hand, it implements the interface to the Altera Avalon bus [1]. The connection of the counter and channel modules mainly consists of routing the `counter` and `counter_changed` signals from the counter unit to the channels. For the bus interface, the module combines channel register interfaces with the same functionality into single memory-mapped registers. More information on the register map can be found in the next chapter. The signals for connection to the Avalon bus are shown in table 2.1. Their widths are based on the current implementation that assumes a word width of 32 bits, requires 5 bits to address all registers of SCCT (see chap. 3), and implements 8 capture/compare channels.

Table 2.1: Avalon bus interface of the SCCT top-level module

Name	Direction	Width (Bits)	Description
clk	in	1	Clock signal
rst	in	1	Reset signal
address	in	5	Read/write address
read	in	1	Read request
readdata	out	32	Data returned for a read request
writedata	in	32	Data to be written
write	in	1	Write request
irq	out	1	Interrupt pending
pins_i	in	8	Input signals, to be connected externally
pins_o	out	8	Output signals, to be connected externally

2.5 Extending SCCT

2.5.1 Adding/Removing Channels

Changing the number of capture/compare channels is performed in the following manner:

1. Change the value of SCCT_N_CHANNELS in `scct_constants.v` (see sect. 4.1).
2. Add/remove channel definitions from `scct.v`. If channel definitions should be added, the `mkch.pl` script can be used to generated these (see sect. 4.2).

When adding channels, keep in mind that the register map of the current implementation can only support up to 16 channels (cf. chap. 3). If you need more channels, you must also adjust the register map.

3 Programming Interface

3.1 Registers

The register map of SCCT is shown in table 3.1. Note that the offsets are shown in terms of words. To calculate the effective address, they must be multiplied by the actual word with. The current implementation is based on words of 32 bit.

Table 3.1: Register map of SCCT

Offset	Name	r/w	Description
0x00	CTR	r	Counter value register
0x04	PSC	rw	Prescaler register
0x08	CTR_IE	rw	Counter interrupt enable register
0x0c	CTR_IS	rw	Counter interrupt status register
0x10	-	-	Reserved
0x14	-	-	Reserved
0x18	-	-	Reserved
0x1c	-	-	Reserved
0x20	CH_MS	rw	Channel mode select register
0x24	CH_AS	rw	Channel action select register
0x28	CH_IE	rw	Channel interrupt enable register
0x2c	CH_IS	rw	Channel interrupt status register
0x30	CH_OCF	w	Channel force action register
0x34	CH_INP	r	Channel input pin status
0x38	CH_OUT	r	Channel output pin status
0x3c	-	-	Reserved
0x40	CCR0	r/w	Channel 0 capture/compare register
0x44	CCR1	r/w	Channel 1 capture/compare register
0x48	CCR2	r/w	Channel 2 capture/compare register
0x4c	CCR3	r/w	Channel 3 capture/compare register
0x50	CCR4	r/w	Channel 4 capture/compare register
0x54	CCR5	r/w	Channel 5 capture/compare register
0x58	CCR6	r/w	Channel 6 capture/compare register
0x5c	CCR7	r/w	Channel 7 capture/compare register
0x60	-	-	Reserved
⋮	⋮	⋮	⋮
0xfc	-	-	Reserved

3.1.1 Counter value register (CTR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTR[31..16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR[15..0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

3.1.2 Prescaler register (PSC)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PSC[31..16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15..0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

3.1.3 Counter interrupt enable register (CTR_IE)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															OIE
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw

OIE: Overflow interrupt enable bit
- **0** : Overflow interrupts are disabled
- **1** : Overflow interrupts are enabled

3.1.4 Counter interrupt status register (CTR_IS)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															OIS
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw

OIS: Overflow interrupt status bit
OIS:
0 : No overflow interrupt pending
1 : Overflow interrupt pending
Write 1 to OIS to reset interrupt. Writing 0 has no effect.

3.1.5 Channel mode select register (CH_MS)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								MS7	MS6	MS5	MS4	MS3	MS2	MS1	MS0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

MSx: Mode selection bit for channel x
0 : Channel is configured as input capture
1 : Channel is configured as output compare

3.1.6 Channel action select register (CH_AS)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH7_IC[1:0]		CH6_IC[1:0]		CH5_IC[1:0]		CH4_IC[1:0]		CH3_IC[1:0]		CH2_IC[1:0]		CH1_IC[1:0]		CH0_IC[1:0]	
CH7_OC[1:0]		CH6_OC[1:0]		CH5_OC[1:0]		CH4_OC[1:0]		CH3_OC[1:0]		CH2_OC[1:0]		CH1_OC[1:0]		CH0_OC[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Depending on the status of the corresponding mode selection bit of the channel, the action select register is either used to configure the input capture reaction (CHx_IC) or the output compare action (CHx_OC) of channel x.

CHx_IC: select type of edge to react on
00 : Ignore all edges
01 : React on rising edge
10 : React on falling edge
11 : React on either edge

CHx_OC: select action on output pin
00 : No OC output action (may still raise an interrupt)
01 : Pull output to high
10 : Pull output to low
11 : Toggle status of output pin

3.1.7 Channel interrupt enable register (CH_IE)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

IEx: Interrupt enable bit for channel x
- **0** : Interrupts from channel x are disabled
- **1** : Interrupts from channel x are enabled

3.1.8 Channel interrupt status register (CH_IS)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

ISx: Interrupt status bit for channel x
- **0** : No capture/compare interrupt in channel x
- **1** : Capture/compare interrupt from channel x is pending

Write 1 to ISx to reset the corresponding channel interrupt. Writing 0 has no effect.

3.1.9 Channel force action register (CH_OCF)

Output pins cannot be set directly, instead a force register is provided. It is used in combination with the `CH_ACT` register. Use this register to set the output pin of an OC channel to a desired level. The behaviour of applying a force action to a channel that is configured as Input-Capture is undefined.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								OCF7	OCF6	OCF5	OCF4	OCF3	OCF2	OCF1	OCF0
-	-	-	-	-	-	-	-	w	w	w	w	w	w	w	w

OCFx: write to this bit to force an output action
- **0** : No action in channel x
- **1** : Force channel x OC action

3.1.10 Channel input pin status (CH_INP)

Use this register to read the current state of the input pins.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								INP7	INP6	INP5	INP4	INP3	INP2	INP1	INP0
-	-	-	-	-	-	-	-	r	r	r	r	r	r	r	r

INPx: state of channel x input pin

3.1.11 Channel output pin status (CH_OUT)

Use this register to read the current state of the output pins.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0
-	-	-	-	-	-	-	-	r	r	r	r	r	r	r	r

OUTx: state of channel x output pin

3.1.12 Channel x capture/compare register (CCRx)

If the channel is configured as input-capture, the `CCRx` register holds the timestamp of the last captured edge. For an output-compare channel, write the time of the output action to the `CCRx` register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCRx[31..16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCRx[15..0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

3.2 Programming Notes

Deactivation of a channel Set the `IENx` bit to 0 and the channel action bits to 00. This can be done in any channel mode. If the value of the `CCRx` register should be retained, set the channel to OC mode (`MSx`=1).

4 Tools and Files

4.1 Verilog Sources

The Verilog source of SCCT are located in the `verilog/` subdirectory.

scct_constants.v contains constants use in the whole implementation, e.g. register widths, flag definitions, etc.

scct_counter.v implements the counter module.

scct_channel.v implements a single capture/compare channel.

scct.v implements the top-level module and the interface to the Altera Avalon bus.

4.2 Tools

mkch.pl is used to generate the channel instantiations for `scct.v`. If you change the channel interface, remember to adjust data section of this script. The output of the script must be copied/pasted into `scct.v`

4.3 Test Environment

Currently, two test cases are implemented for SCCT:

test_channel.v Test case for a single channel.

test_scct.v Test case for the whole SCCT module.

They can be found in the `test/` subdirectory of the package.

Test cases can be built using the `Makefile`. This requires that the `iverilog` package[1] is installed. Additionally, the `Makefile` requires that the variable `${CASE}` is set to either `channel` or `scct`. The following functions are supported by the Makefile:

- Building:

```
$ CASE=${CASE} make
```

- Executing (automatically builds the project):

[1] `http://iverilog.icarus.com/`

```
$ CASE=${CASE} make run
```

Exit the simulation by typing `finish`. The simulation trace is written to the file specified by the `$(DUMP)` variable in the Makefile.

- Viewing (run the simulation before!):

```
$ CASE=${CASE} make view
```

Requires that `gtkwave`[2] is installed on the system.

If you want to add further test cases, name them `test\_TESTCASENAME.v`. Calls to `make` then have the following form:

```
$ CASE=TESTCASENAME make ...
```

4.4 C Header

A C header file for SCCT can be found in the `include/` directory. The file contains definitions for the register offsets defined in the memory map (see table 3.1) and some macros to calculate actual bitmasks.

[2] http://gtkwave.sourceforge.net/

5 Bibliography

[1] Altera Corporation, 101 Innovation Drive, San Jose, CA 95134. *Avalon Interface Specifications*, 2015.