Software usage information

Generation of PWM waveforms using the SCT

Rev. 1.2 — 07 August 2012

Document information

Info	Content
Keywords	SCT, PWM, Phase, Push Pull, Dead Band, Pulse Skipping
Abstract	This application notes describes several examples which show how to configure the SCT for generating PWM waveforms

Revision history

Rev	Date	Description
1.0	19 July 2012	First release
1.1	23 July 2012	Added support for variable frequency pwm
1.2	07 Aug 2012	Added support for complementary pulse skipping waveform

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

This application note shows how to use the SCT to generate several PWM waveform types.

The software examples provide an API interface which can be used to configure the peripheral to support the following modes:

- Single phase
- Single phase with complementary outputs
- Push pull
- Push pull with complementary outputs
- Pulse skipping
- Pulse skipping with complementary outputs
- Variable frequency with fixed duty cycle

The following sections detail the configuration options and the signals generated by the SCT. All the configuration options are related to hardware features and settings which are described in the device user manual.

2. SCT configuration API

The test example provides several predefined test configurations the user can modify.

The following PWM Types are supported

Table 1. Supported PWM Types

Functionality	PwmType
Single phase	SINGLE_PHASE
Single phase with complementary outputs	COMP_SINGLE_PHASE
Push pull	PUSH_PULL
Push pull with complementary outputs	COMP_PUSH_PULL
Pulse skipping	PULSE_SKIPPING
Pulse skipping with complementary outputs	COMP_PULSE_SKIPPING
Variable frequency with fixed duty cycle	VARIABLE_F_FIXED_DC

To initialize the SCT, the following call needs to be made first:

```
PWM_STATUS SCT_pwmInit(PwmGlobalParams *config)
```

This function call returns a status value which can be checked against the following values:

```
PWM_OK
PWM_INVALID_CONFIGURATION
PWM_INVALID_COMMAND
PWM_INVALID_BASE_FREQUENCY
PWM_INVALID_TIMING
PWM_INVALID_DC
PWM_INVALID_FREQ
PWM_INVALID_STATUS
```

The `PwmGlobalParams` configuration structure includes the following fields:

a) `PwmClockBase clock`

The clock source provided to the SCT can be set to `CONFIG_CLKMOD_INTERNAL` (internal clock), `CONFIG_SAMPLED_HIGHPERF` (high performance sampled external clock), `CONFIG_SAMPLED_LOWPOWER` (low power sampled external clock), `CONFIG_CLKSEL_INPUT` (asynchronous external clock)

b) `PwmClockInput clockIn`

In case an external clock reference shall be used, the `clockIn` parameter defines the sampled input pin

c) `PwmClockFrequency pwmFrequency`

This defines the maximum frequency that the generated PWM should run to. It basically defines which temporal resolution should be achieved by any PWM signal when switched

d) `SctClockFrequency sctFrequency`

Defines the frequency of the clock signal provided to the SCT peripheral block. It is required in order to define internally the SCT counter pre-scale value required, in relation to the desired `pwmFrequency`

To configure the SCT, the following API can be used:

```
PWM_STATUS SCT_pwmOpen(PwmType mode, const void* config)
```

The desired PwmType mode can be chosen among of the items listen in Table 1 above.

The specific configuration parameter depends on the PwmType chosen, and is detailed in the following paragraphs. Note: after opening the SCT in a specific emulation mode, the peripheral is kept halted and needs to be started by the application using the `SCT_pwmIoctl()` function call.

To change dynamically the SCT behavior, some of the configuration parameters, or to start and stop the SCT, use the following API:

```
PWM_STATUS SCT_pwmIoctl(const PwmIoctl cmd, void* param)
```

The list of available commands is shown in Table 2 below

Table 2. List of supported Ioctl commands

Command	Parameter	Effect
CMD_EMPTY	-	None
GET_PWM_FREQUENCY	Pointer to PwmClockFrequency variable	Returns the current clock base for the SCT
START_PWM	-	Start generating the PWM signals
SET_DC	Pointer to new duty cycle value	Change the duty cycle
SET_FREQ	Pointer to new frequency value	Change the period frequency
ENABLE_INT	Interrupt mask	Activates the event generated interrups
SYNC_STOP_PWM	-	Stops the SCT counter at the next period boundary
STOP_PWM	-	Stops the SCT counter

Command	Parameter	Effect
		immediately
SYNC_HALT_PWM	-	Halts the SCT counter at the next period boundary
HALT_PWM	-	Halts the SCT counter immediately
BLOCK_RELOAD	-	Stops reloading of match registers on limit events
ENABLE_RELOAD	-	Enables reloading of match registers on limit events

To stop the emulated peripheral, call the following API:

```
void SCT_pwmClose(void)
```

This function resets the SCT and brings the peripheral again in its default state. All previous configuration settings are lost.

3. Single phase

This is the most simple of the PWM waveforms.

The configuration structure to be used within the `SCT_pwmOpen()` function is as follows:

```
typedef struct PWM_SinglePhase {
  uint16_t    outputMask;
  uint16_t    interruptMask;
  uint16_t    dutyCycle;
  uint32_t    period;
  PWM_polaritypolarity;
  intCallback_t  risingEdgeInt;
  intCallback_t  fallingEdgeInt;
  intCallback_t  periodInt;

} PWM_SinglePhase;
```

a) outputMask

defines which SCT outputs shall be driven and is a bit mask, bit zero relates to SCT_OUT0 and bit 15 to SCT_OUT_16

b) interruptMask

can be set to an OR'ed combination of the following values: RISING_INT, FALLING_INT and PERIOD_INT. Each of these is related to the event setting the PWM pulse, clearing the PWM pulse, or marking the period boundary

c) duty cycle

is represented 'per mil', so a value of 1000 represents 100% duty cycle, 500 represents 50%, 101 represents 10,1% and so on

d) period

specifies the base frequency in Herz at which the PWM period is generated, i.e. the period frequency

e) polarity

defines if the PWM signal should start active high or low at the beginning of the period

f) risingEdgeInt, fallingEdgeInt, periodInt

interrupt callback functions related to the event interrupts. In case not used, define them as empty functions.

Otherwise, these functions are executed within the SCT Interrupt context when the rising edge, falling edge or period event generates the interrupt.

Note: the interrupt does not need to be acknowledged (quit) within the callback function since it is already done within the SCT interrupt routine

Below a snapshot of how the generated signals could look like:

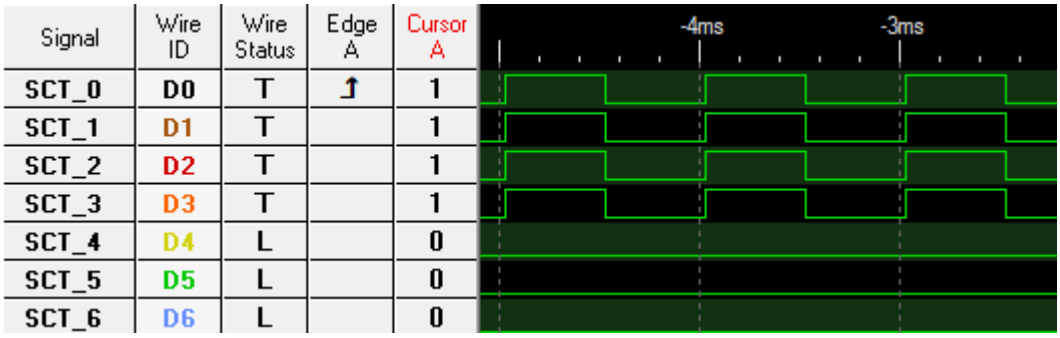

Fig 1. Example of PWM single phase generation

In this example, SCT0 to SCT3 are used.

The same PWM waveform is presented on all defined outputs.

4. Single phase with complementary outputs

This configuration provides a similar setup like for the simple single phase, but with additional complementary signals which are suitable to drive a power bridge, for example.

The configuration structure to be used within the `SCT_pwmOpen()` function is similar to the single phase setup, with some additional items:

```
typedef struct PWM_outputPair {
   SCT_OUT directOutput;
   SCT_OUT complementaryOutput;
} PWM_outputPair;

typedef struct PWM_phases {
   uint8_t        numPhases;
   PWM_outputPair*  phasePairs;
} PWM_phases;

typedef struct PWM_ComplementarySinglePhase {
   PWM_phases     pwmOutputs;
   uint16_t       interruptMask;
   uint16_t       dutyCycle;
   uint32_t       period;
   PWM_polarity   polarity;
   uint16_t       risingEdgeDeadband;
   uint16_t       fallingEdgeDeadband;
   intCallback_t  risingEdgeInt;
   intCallback_t  fallingEdgeInt;
   intCallback_t  periodInt;
} PWM_ComplementarySinglePhase;
```

The new items, compared to the single phase configuration are:

a) pwmOutputs

   defines which outputs shall be driven on each direct and complementary signal. It is a structure including the number of signal pairs, and a pointer to an array of substructures defining which outputs shall be considered in pairs

b) risingEdgeDeadBand, fallingEdgeDeadBand

risingEdgeDeadBand defines the amount of time which is inserted between the direct output being switched on and its complementary output being switched off, when the rising edge event is fired (beginning of the period)

fallingEdgeDeadBand defines the amount of time in timer ticks which is inserted between the direct output being switched on and its complementary output being switched off, when the falling edge event is fired

Note: both are expressed in timer ticks, so have to be calculated referring to the PWM frequency period (1/period)

Below a snapshot of how the generated signals could look like:

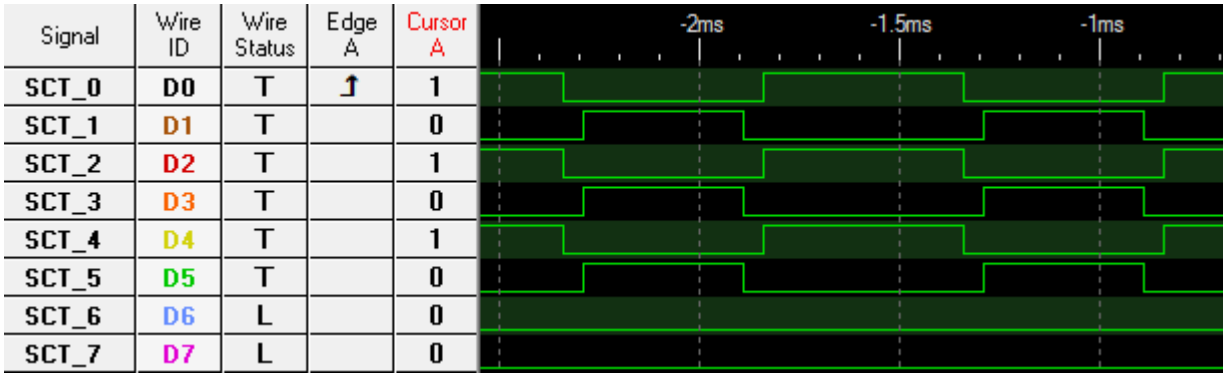

Fig 2. Example of PWM complementary single phase generation

In this example, SCT0, SCT2, SCT4 are the direct outputs, and SCT1, SCt3, SCT5 are the complementary outputs.

The same PWM waveform is presented on all direct or complementary outputs.

5. Push pull

This configuration provides a setup for which a set of outputs is driven during odd numbered periods, and a different set of outputs is driven during even numbered periods.

The configuration structure to be used within the `SCT_pwmOpen()` function is similar to the single phase setup, with some additional items:

```
typedef struct PWM_PushPull {
  uint16_t    outputMaskOdd;
  uint16_t    outputMaskEven;
  SCT_OUT     oddPeriod;
  uint16_t    interruptMask;
  uint16_t    dutyCycle;
  uint32_t    period;
  PWM_polarity   polarity;
  intCallback_t  risingEdgeInt;
  intCallback_t  fallingEdgeInt;
  intCallback_t  periodInt;
} PWM_PushPull;
```

a) outputMaskOdd, outputMaskEven

define which outputs shall be driven and is a bit mask, bit zero relates to SCT_OUT0 and bit 15 to SCT_OUT_16. Each output in outputMaskOdd is driven during odd periods, each output in outputMaskEven is driven in even periods

b) oddPeriod

internal only SCT output signal which is used to track in hardware the even and odd periods. One SCT output signal needs to be reserved for this purpose and cannot be used as PWM output

Below a snapshot of how the generated signals could look like:

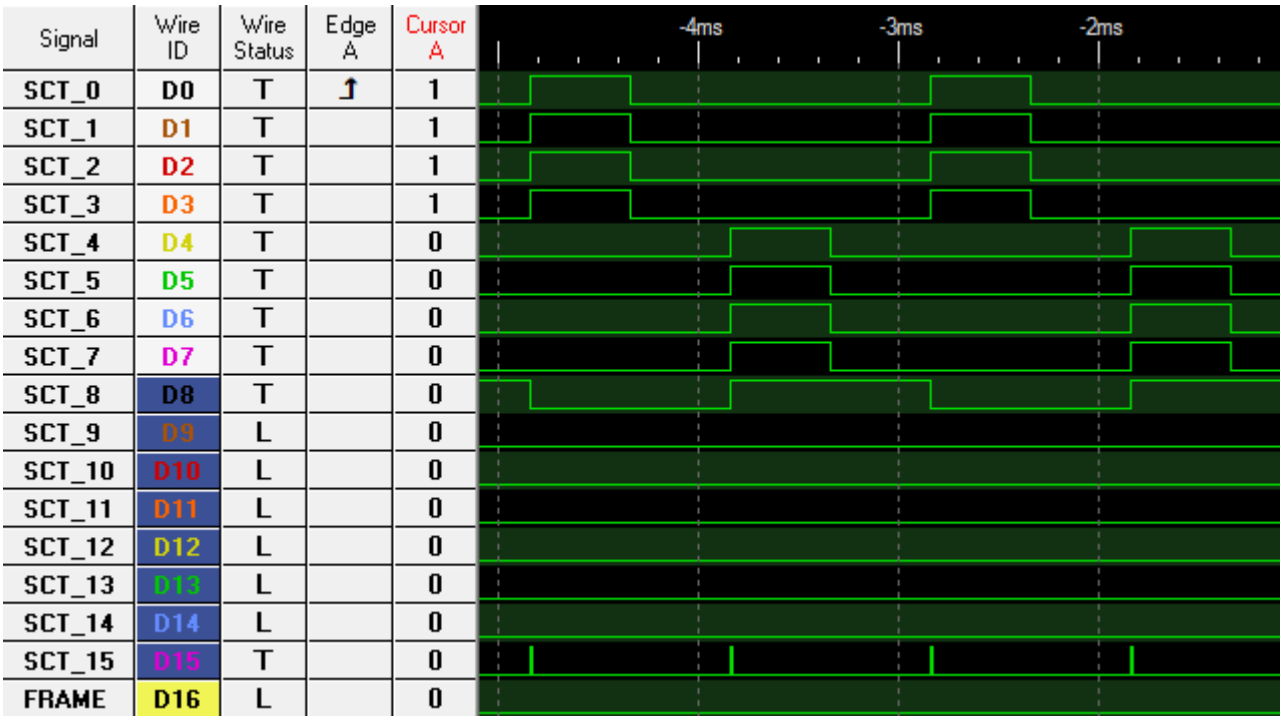

Fig 3. Example of PWM push pull generation

In this example configuration, SCT outputs 0 to 3 are configured to be active during the even periods, and SCT outputs 4 to 7 are configured to be active during the odd periods.

SCT 8 is used as debug output, showing the internal signal used to track the odd and even periods. An odd period is considered to be active when the output is high.

SCT output 15 is also used as a debug output which marks the PWM period boundary

6. Push pull with complementary outputs

This configuration provides a similar setup like for the push pull configuration, but with additional complementary signals.

The configuration structure to be used within the SCT_pwmOpen() function is similar to the single phase setup, with some additional items:

```
typedef struct PWM_pushPullOutputPair {
   SCT_OUT directOutput;
   SCT_OUT complementaryOutput;
} PWM_pushPullOutputPair;


typedef struct PWM_pushPullPhases {
   uint8_t          numPhases;
   PWM_pushPullOutputPair* phasePairs;
} PWM_pushPullPhases;


typedef struct PWM_ComplementaryPushPull {


   PWM_pushPullPhases    pwmPhasesOdd;
   PWM_pushPullPhases    pwmPhasesEven;
   SCT_OUT     oddPeriod;
   uint16_t       interruptMask;
   uint16_t    dutyCycle;
   uint32_t    period;
   uint16_t    risingEdgeDeadband;
   uint16_t    fallingEdgeDeadband;
   PWM_polaritypolarity;
   intCallback_t  risingEdgeInt;
   intCallback_t  fallingEdgeInt;
   intCallback_t  periodInt;


} PWM_ComplementaryPushPull;
```

The new items, compared to the single phase configuration are:

a) PWM_pushPullPhases pwmPhasesOdd

defines which outputs shall be driven on each direct and complementary signal during an odd period.

It is a structure including the number of signal pairs, and a pointer to an array

of substructures defining which outputs shall be considered in pairs

b) `PWM_pushPullPhases  pwmPhasesEven`

defines which outputs shall be driven on each direct and complementary signal during an even period, like for `pwmPhasesOdd`.

c) risingEdgeDeadBand, fallingEdgeDeadBand

risingEdgeDeadBand defines the amount of time which is inserted between the direct output being switched on and its complementary output being switched off, when the rising edge event is fired (beginning of the period)

fallingEdgeDeadBand defines the amount of time in timer ticks which is inserted between the direct output being switched on and its complementary output being switched off, when the falling edge event is fired

Note: both are expressed in timer ticks, so have to be calculated referring to the PWM frequency period (1/period)

Below a snapshot of how the generated signals could look like:

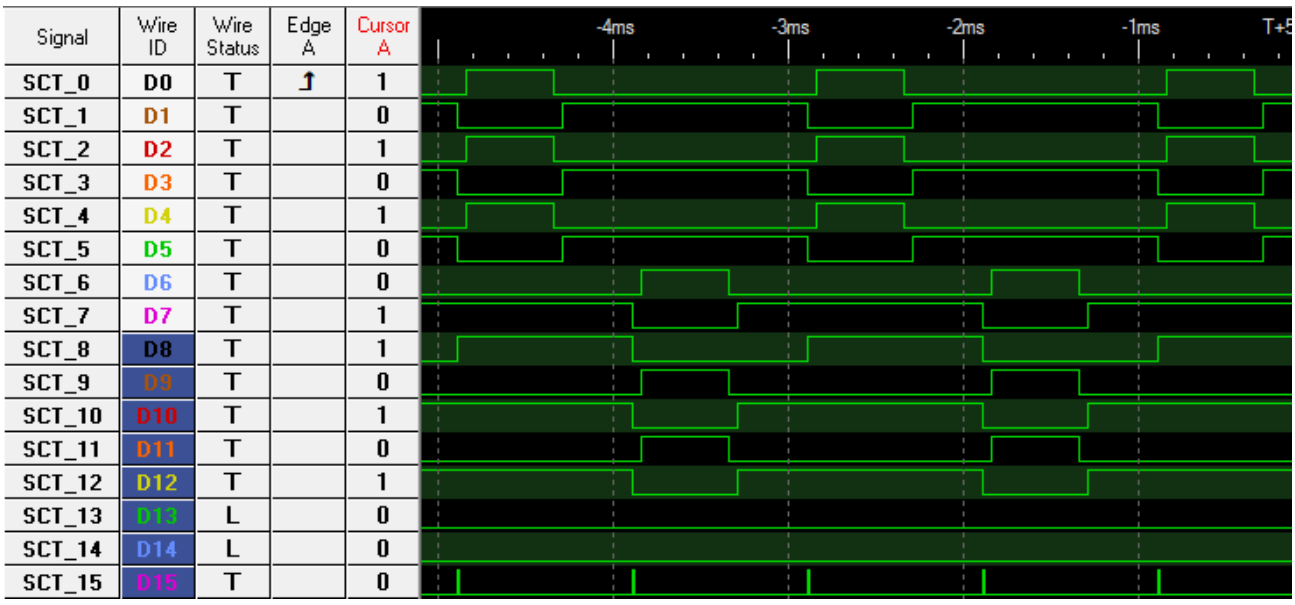

Fig 4. Example of PWM complementary push pull generation

In this example configuration, SCT outputs 0,2,4 and 1,3,5 are configured to be active during the even periods, and SCT outputs 6,9,11 and 7,10,12 are configured to be active during the odd periods.

SCT 8 is used as debug output, showing the internal signal used to track the odd and even periods. An odd period is considered to be active when the output is high.

SCT output 15 is also used as a debug output which marks the PWM period boundary

7. Pulse skipping

This configuration provides a similar setup like for the single phase configuration, but with an additional pulse skip input signal.

This asynchronous input signal is evaluated at every period boundary.

If the signal is sampled high, the PWM outputs are active in the successive period.

If the signal is sampled low, the PWM outputs are left idle and are inactive.

The configuration structure to be used within the `SCT_pwmOpen()` function is similar to the single phase setup, with some additional items:

```
typedef struct PWM_PulseSkipping {
  uint16_t    outputMask;
  SCT_IN      asyncEnableInput;
  uint16_t      interruptMask;
  uint16_t    dutyCycle;
  uint32_t    period;
  PWM_polaritypolarity;
  intCallback_t  risingEdgeInt;
  intCallback_t  fallingEdgeInt;
  intCallback_t  periodInt;
} PWM_PulseSkipping;
```

The new items, compared to the single phase configuration are:

a) asyncEnableInput

defines which input signal is used to control the skipping of the PWM signals within a specific period

Below a snapshot of how the generated signals could look like:

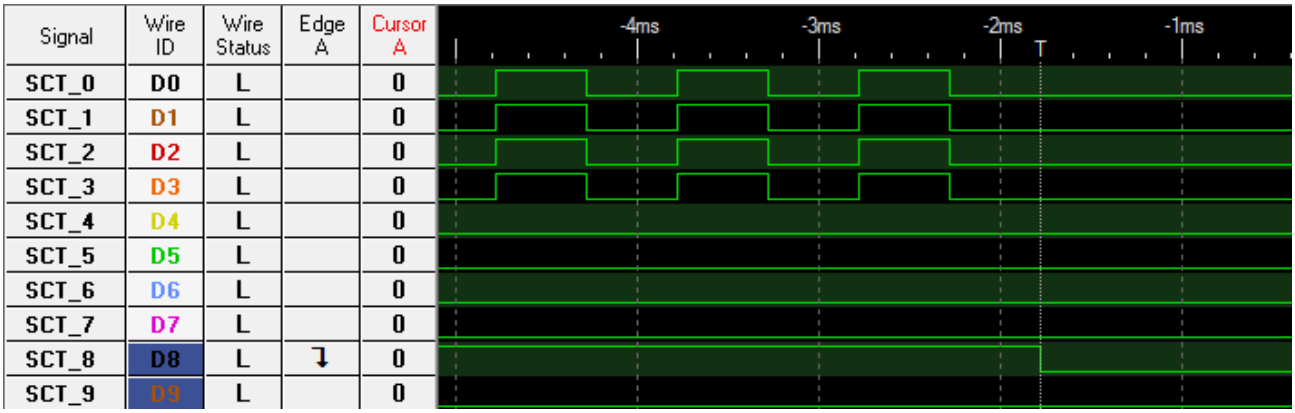

Fig 5. Example of PWM pulse skipping generation – begin of skip sequence

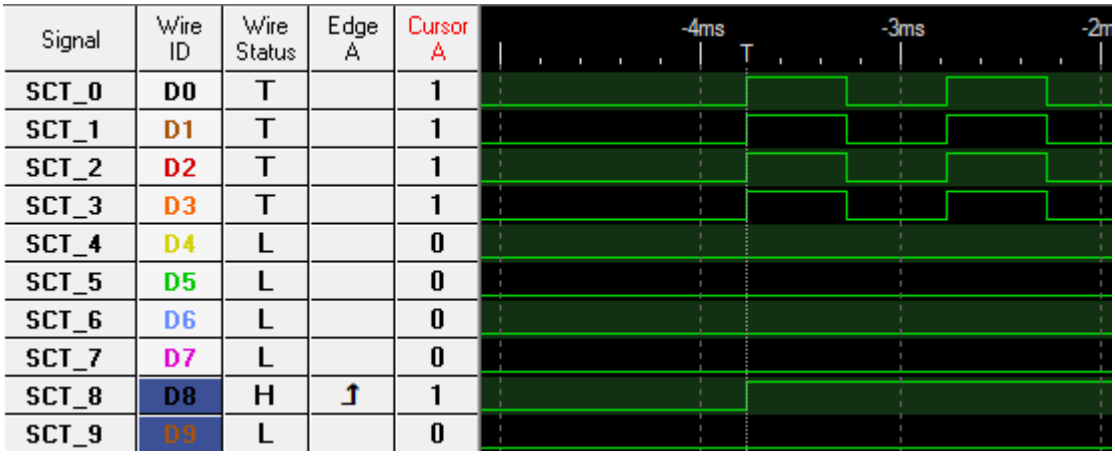

Fig 6. Example of PWM pulse skipping generation – end of skip sequence

In this example configuration, SCT outputs 0,1,2,3 are configured to be active when the asynchronous asyncEnableInput signal is sampled low

SCT output 8 is a debug signal, which is set low when the asyncEnableInput is detected low at the beginning of a skip sequence, and set high when the asyncEnableInput is detected high, so it tracks the status of the asyncEnableInput in correspondence to a period boundary

8. Pulse skipping with complementary outputs

This configuration provides a similar setup like for the pulse skipping configuration, but with additional complementary outputs.

The configuration structure to be used within the `SCT_pwmOpen()` function is similar to the pulse skipping setup, with some additional items:

```
  typedef struct PWM_pulseSkipOutputPair {
    SCT_OUT directOutput;
    SCT_OUT complementaryOutput;
  } PWM_pulseSkipOutputPair;

  typedef struct PWM_pskipPhases {
    uint8_t                   numPhases;
    PWM_pulseSkipOutputPair*  phasePairs;
  } PWM_pskipPhases;

typedef struct PWM_ComplementaryPulseSkipping {
  PWM_pskipPhases  pwmOutputs;
  SCT_IN           asyncEnableInput;
  uint16_t         interruptMask;
  uint16_t         dutyCycle;
  uint32_t         period;
  uint16_t         risingEdgeDeadband;
  uint16_t         fallingEdgeDeadband;
  PWM_polarity     polarity;
  intCallback_t    risingEdgeInt;
  intCallback_t    fallingEdgeInt;
  intCallback_t    periodInt;
} PWM_ComplementaryPulseSkipping;
```

The new items, compared to the pulse skipping configuration are:

a) pwmOutputs

defines which outputs shall be driven on each direct and complementary signal. It is a structure including the number of signal pairs, and a pointer to an array of substructures defining which outputs shall be considered in pairs

9. Variable frequency – Fixed duty cycle

This configuration provides a similar setup like for the single phase configuration, but with a fixed duty cycle value. The user is allowed to change the PWM frequency by means of the `pwmIoctl` api function. The duty cycle is preserved after the change, assuming the combination of the new frequency and the constant duty cycle value are possible given the reference base PWM frequency used by the SCT (specified in the global settings).

The configuration structure to be used within the `SCT_pwmOpen()` function is similar to the single phase setup:

```
typedef struct PWM_VarFrequency {
  uint16_t      outputMask;
  uint16_t      interruptMask;
  const uint16_t dutyCycle;
  uint32_t      period;
  PWM_polarity  polarity;
  intCallback_t risingEdgeInt;
  intCallback_t fallingEdgeInt;
  intCallback_t periodInt;
} PWM_VarFrequency;
```

10. Examples and configuration software

The configuration software can be downloaded from this site:

http://git.lpcware.com/lpc43xx.git

Within the directory located at Examples\SCT\Sct_pwm, open the Keil\sct_pwm.uvproj project file and rebuild the application to test.

The main file, SCT_pwm_test.c includes several predefined configurations which can be chosen for the tests.

11. Legal information

11.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

11.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

11.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

11.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

11.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

<Name> — is a trademark of NXP B.V.

12. List of figures

13. List of tables

14. Contents

For more information, visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 07 August 2012
Document identifier: