AN11536

Using the SCTimer/PWM for capacitive touch buttons

Rev. 1 — 14 April 2014 Application note

Document information

Info	Content
Keywords	LPC800, SCTimer/PWM, Analog Comparator, Capacitive Touch, Switch Matrix
Abstract	This application note describes a simple capacitive touch sensing method using the analog comparator and State Configurable Timer (SCTimer/PWM)

Revision history

Rev	Date	Description
1	20140414	Initial version.

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

Capacitive touch buttons are commonly used in embedded systems due to their low cost, high reliability, and simplicity compared to mechanical switches. The capacitive button can be constructed using a pad on a PCB.

This application note describes a simple capacitive touch sensing method using the analog comparator and State Configurable Timer (SCTimer/PWM) peripherals that are included in the LPC81xM family of 32-bit microcontrollers.

2. Theory of operation

2.1 Comparator used to implement a relaxation oscillator

A comparator can be used to construct a relaxation oscillator as shown in Fig 1.

If we assume that the positive input is set at a voltage of Vth, and initially, the voltage across C1 is zero, the output of the comparator will be high. This causes the capacitor to charge through resistor R1 until it reaches voltage Vth.

When the negative input voltage is greater than Vth, the output of the comparator will go low, discharging capacitor C1 through R1. This process will occur continuously creating an oscillator.

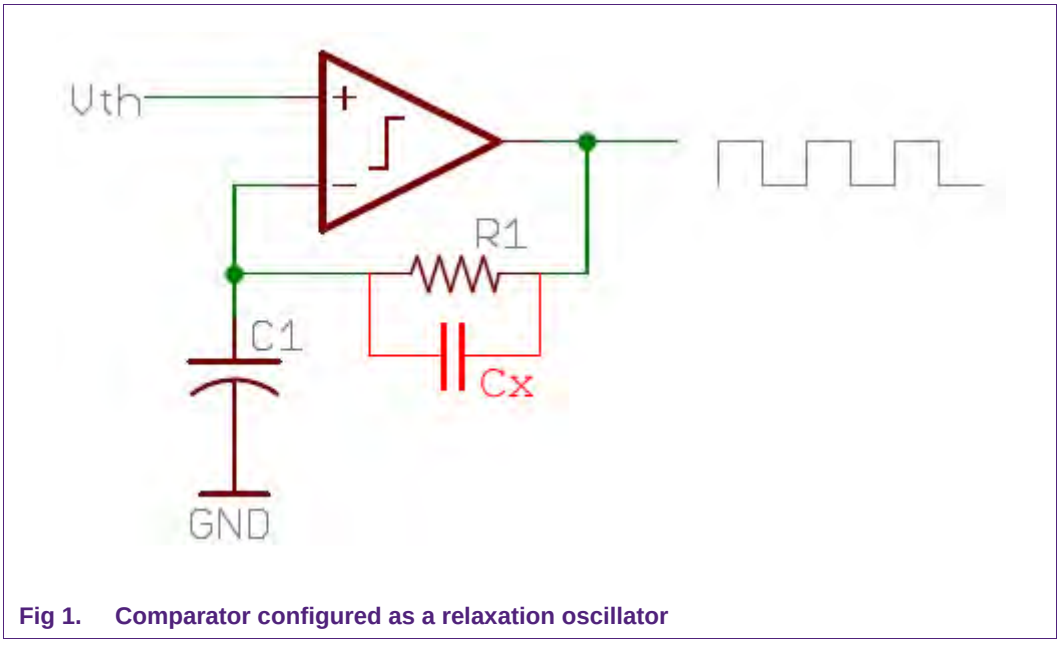


Fig 1. Comparator configured as a relaxation oscillator

Rather than using an actual capacitor for C1, it can also be created using a pad on a PCB. When there is nothing touching the PCB pad, the capacitance of the PCB pad and resistor R1 will create an oscillator with a specific frequency (see Fig 3).

When a finger is placed on the PCB pad, it will increase the capacitance of the pad, causing the oscillation frequency to decrease (see Fig 4). By measuring the period of the comparator output, we can determine whether a finger is touching the PCB or not.

A parasitic capacitance Cx on R1 or on the board layout will cause the small step in the R1 C1 charge / discharge waveform (see Fig 2).

Changing the comparator input hysteresis between 5 mV and 20 mV will only affect the timing by about 70 % and can be calculated into the SCTimer/PWM setup data. It is also advised to use the 20 mV threshold to provide better noise immunity.

Note that the capacitance of the scope probe (10 pf) on C1 in Fig 2 will change the RC timing by almost 100 %.

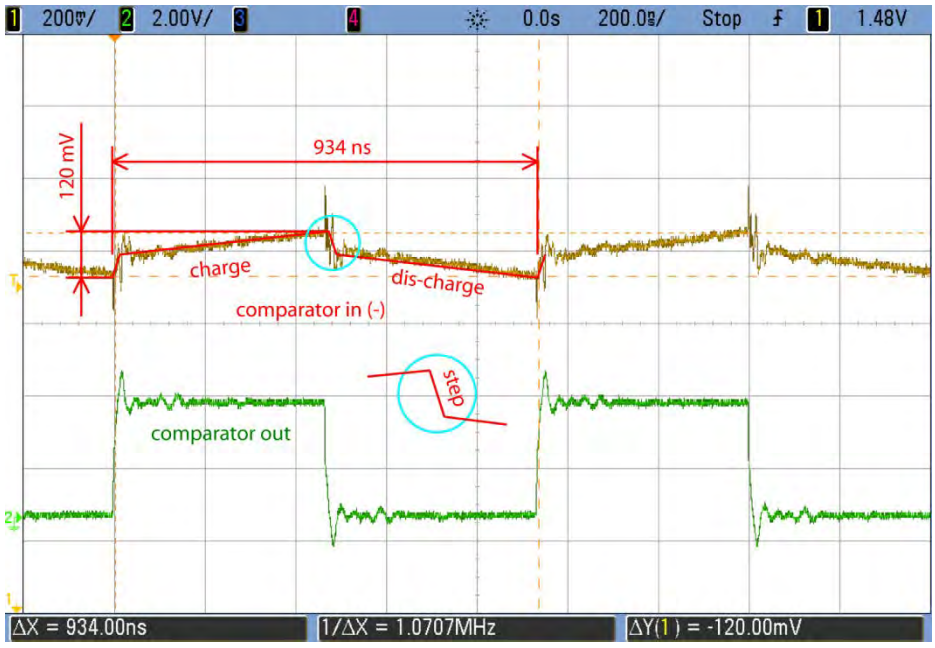


Fig 2. Oscilloscope trace showing R1 C1 oscillator waveform without button press (top trace)

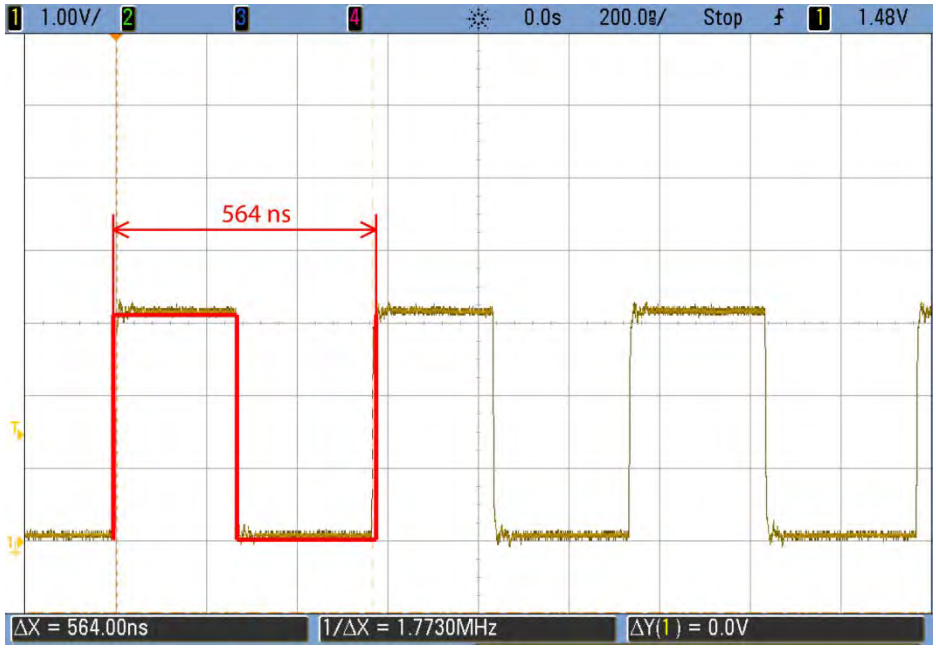


Fig 3. Oscilloscope trace showing output of oscillator without button press

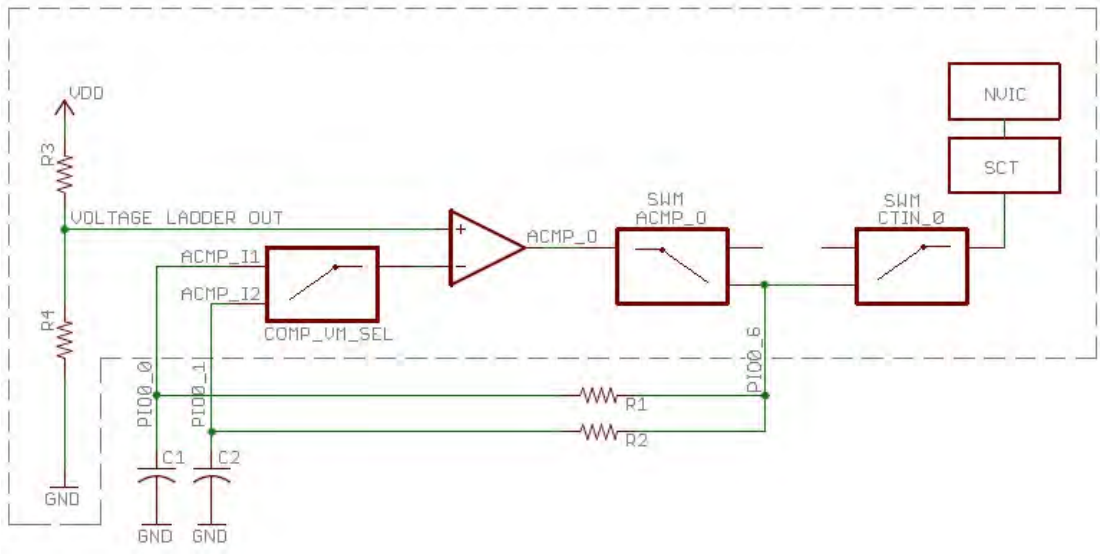
Fig 4. Oscilloscope trace showing output of oscillator with button press

3. Capacitive touch buttons using the LPC81xM

In order to create a solution with two capacitive buttons, with a single analog comparator input, we will utilize the following features of the LPC81xM microcontroller:

1. Analog comparator
2. SCTimer/PWM
3. Switch Matrix

The block diagram of the solution is shown in Fig 5. The only external components required are two resistors R1, R2 and two PCB pads that function as capacitors C1 and C2.



Note: Area inside dashed lines are internal to the LPC81xM

Fig 5. Capacitive touch block diagram

3.1 Switch matrix

The switch matrix can dynamically change how the peripherals in microcontrollers are connected to the pins of the device package. The switch matrix is used on pins PIO0_6 for ACMP_0 and CTIN_0.

COMP_VM_SEL has a fixed pin assignment: PIO0_0 is ACMP_I1, PIO0_1 is ACMP_I2, and is actually part of the analog comparator and selects which input is tied to the negative input of the comparator.

The analog comparator output ACMP_O is connected to PIO0_6 through the switch matrix to feed both buttons, C1 and C2 through R1 and R2. The switch matrix also connects the SCT (CTIN_0) input to the same PIO0_6, so the output of the relaxation oscillator is fed into the SCTimer/PWM. After a scanning period (completion of one flow chart loop) has elapsed, the program then switches the comparator COMP_VM_SEL input so that the negative input of the comparator is connected to the other comparator input of the two buttons, C1 PIO0_0 or C2 PIO0_1.

3.2 Analog comparator

A block diagram of the analog comparator is shown in Fig 6. The features used are highlighted in red. The voltage ladder output is used for the positive input of the comparator. The voltage ladder is powered by V_{DD}.

The negative input of the comparator is selected by the source code to be either ACMP_I1 or ACMP_I2. Button 1 is connected to ACMP_I1 while button 2 is connected to ACMP_I2. Note that the comparator inputs are located on fixed pins PIO0_0 and PIO0_1.

The output of the comparator ACMP_O is connected to the pins of the microcontroller. The switch matrix is used to select which pin the output of the comparator is connected to. See Section 3.1 for details of the switch matrix configuration.

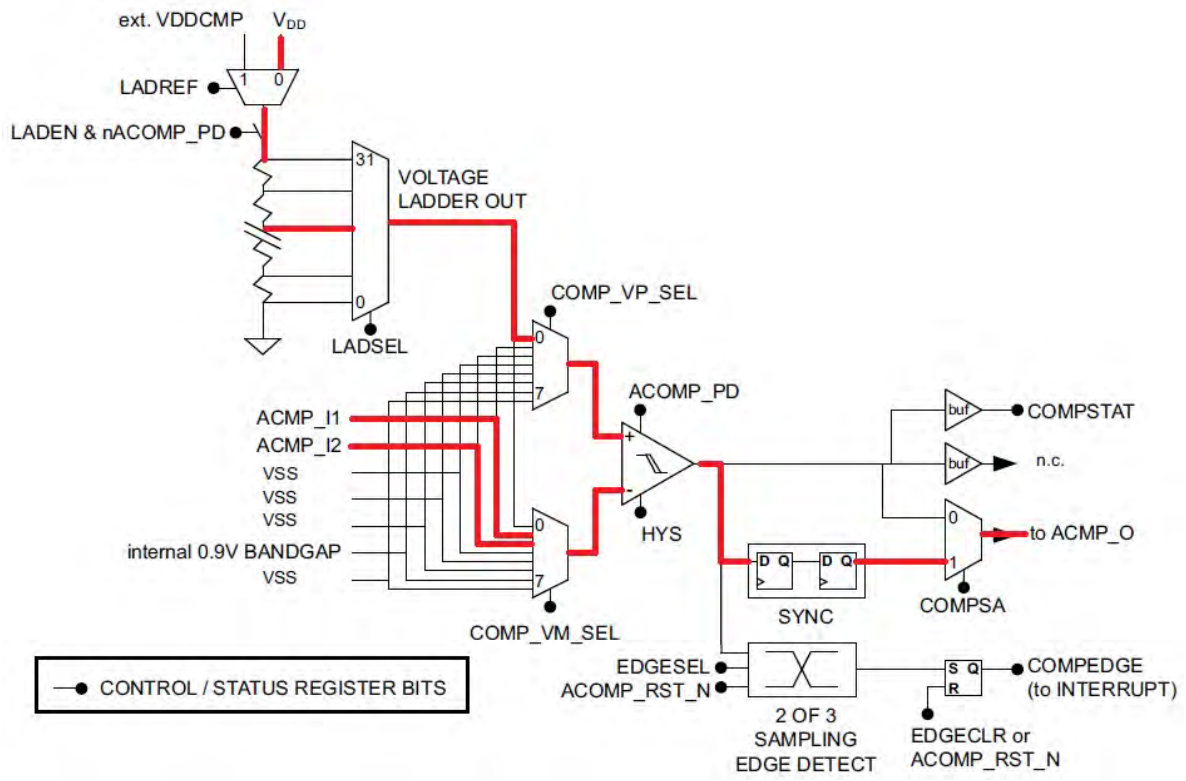


Fig 6. Analog comparator

3.3 SCTimer/PWM

The SCTimer/PWM is a peripheral that is unique to NXP Semiconductors. It can operate like most traditional timers, but also adds a state machine to give it a higher degree of configurability and control. This allows the SCTimer/PWM to be configured as multiple PWMs, a PWM with dead-time control, a PWM with reset capability, as well as other configurations that cannot be duplicated with traditional timers. Once the SCTimer/PWM has been configured, it can run totally autonomously from the microcontroller core.

The SCTimer/PWM implemented in the LPC812M has the following features:

- Two 16-bit counters or one 32-bit counter.
- Counters clocked by bus clock or selected input.
- Up counters or up-down counters.
- State variable allows sequencing across multiple counter cycles.
- The following conditions define an event: a counter match condition, an input (or output) condition, a combination of a match and/or an input/output condition in a specified state, and the count direction.
  - Events control outputs, interrupts, and the SCTimer/PWM states.
  - Match register 0 can be used as an automatic limit.
  - In bidirectional mode, events can be enabled based on the count direction.
  - Match events can be held until another qualifying event occurs.
- Selected events can limit, halt, start, or stop a counter.
- Supports:
  - Four inputs/four outputs
  - Five match/capture registers
  - Six events
  - Two states

4. Schematics and PCB

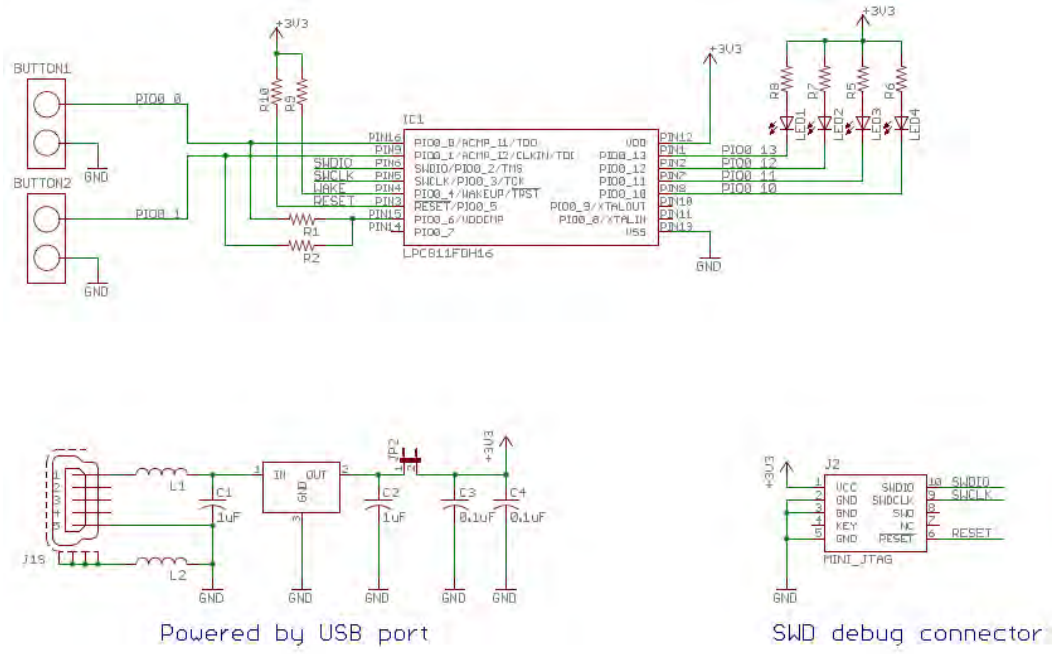


Fig 7. Capacitive touch button schematics

Note: The PCB schematic had a small change which is reflected in the PCB layout pictures below as a yellow colored jumper connection. The purpose was to connect R2 to P0.6. The connection from R2 to P0.9 is also no longer required. This last minute change is not incorporated in the layout files and Gerber files.

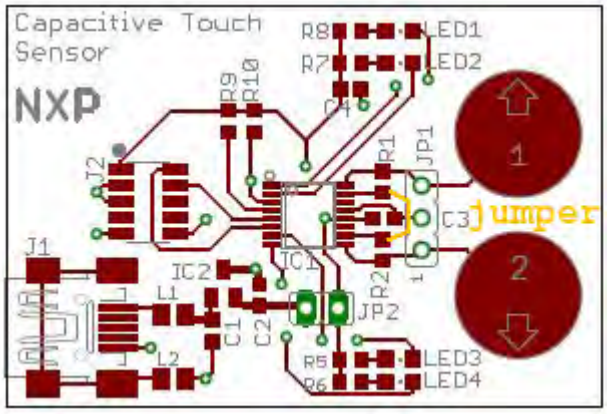


Fig 8. PCB top

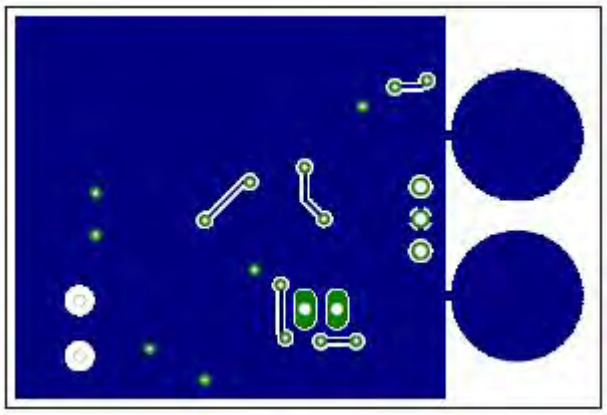


Fig 9. PCB bottom

5. Using the SCTimer/PWM

Fig 10 shows the Red State diagram used to generate the SCTimer/PWM initialization code. The following SCTimer/PWM resources are used:

- One SCTimer/PWM input (total of four available).
- One SCTimer/PWM output (total of four available).
- Three 16-bit match registers.
- One 16-bit capture register.
- Five events (six available).

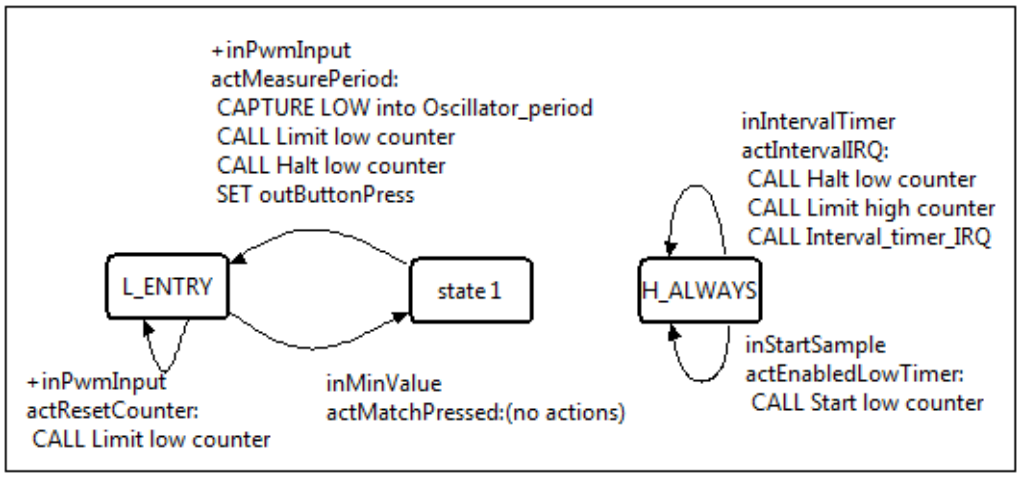


Fig 10. Red State diagram of the capacitive touch button application

1. Initially, the low 16-bit counter is stopped (disabled) while the high 16-bit counter is running. Since it takes some time for the interrupt service routine to switch channels and then for the relaxation oscillator to start-up and stabilize, a “start_sample” delay is incorporated. When the “start_sample” match occurs, it will start the low counter.
2. Now that the low timer is running, it will begin looking for a rising edge on the “pwm_input”. Every time a rising edge occurs on the pwm_input, it will limit (or reset) the low counter. If a second rising edge is detected before the match_pressed period has elapsed, the timer will once again reset, and it will wait for the next rising edge. The LOW counter will stay in the L_ENTRY state until the period of the signal exceeds “match_pressed” so a match event occurs, and the low timer will transition to state 1. This will only occur if the period of the pwm_input is greater than the interval defined by “match_pressed”. If a button is pressed, the period of the oscillator should allow it to exceed the match_pressed period. The LOW timer will wait in “state 1” until another rising edge is detected. If a rising edge is detected, the SCTimer/PWM will capture the count into “Oscillator_period”. It will then limit (reset) the LOW counter, and HALT the LOW counter so no further sampling occurs. It will be back in “L_ENTRY” but it will not be operating since it has been halted.
3. The HIGH counter will continue to count until it matches “interval_timer” which is set to 2 ms (SystemCoreClock/500=60000 clock ticks). At this point, it will HALT the LOW timer (if it is not already HALTed), limit the HIGH timer, and then call the SCTimer/PWM interrupt Interval_IRQ.

The LOW counter is now halted (idle), while the HIGH counter once again begins counting towards the “start_sample”.

4. The SCTimer/PWM interrupt Interval_IRQ will switch the hardware connections between the two RC touchpads and unhalt counter_L and stop counter_L within one write instruction. Reset state_L since it could be in either state at the time of the IRQ and “ack” the interrupt.

Fig 11 outlines the flow of events and two states the SCT is operating on.

Fig 12 shows the LPCXpresso Red State generated SCTimer/PWM initialization function code.

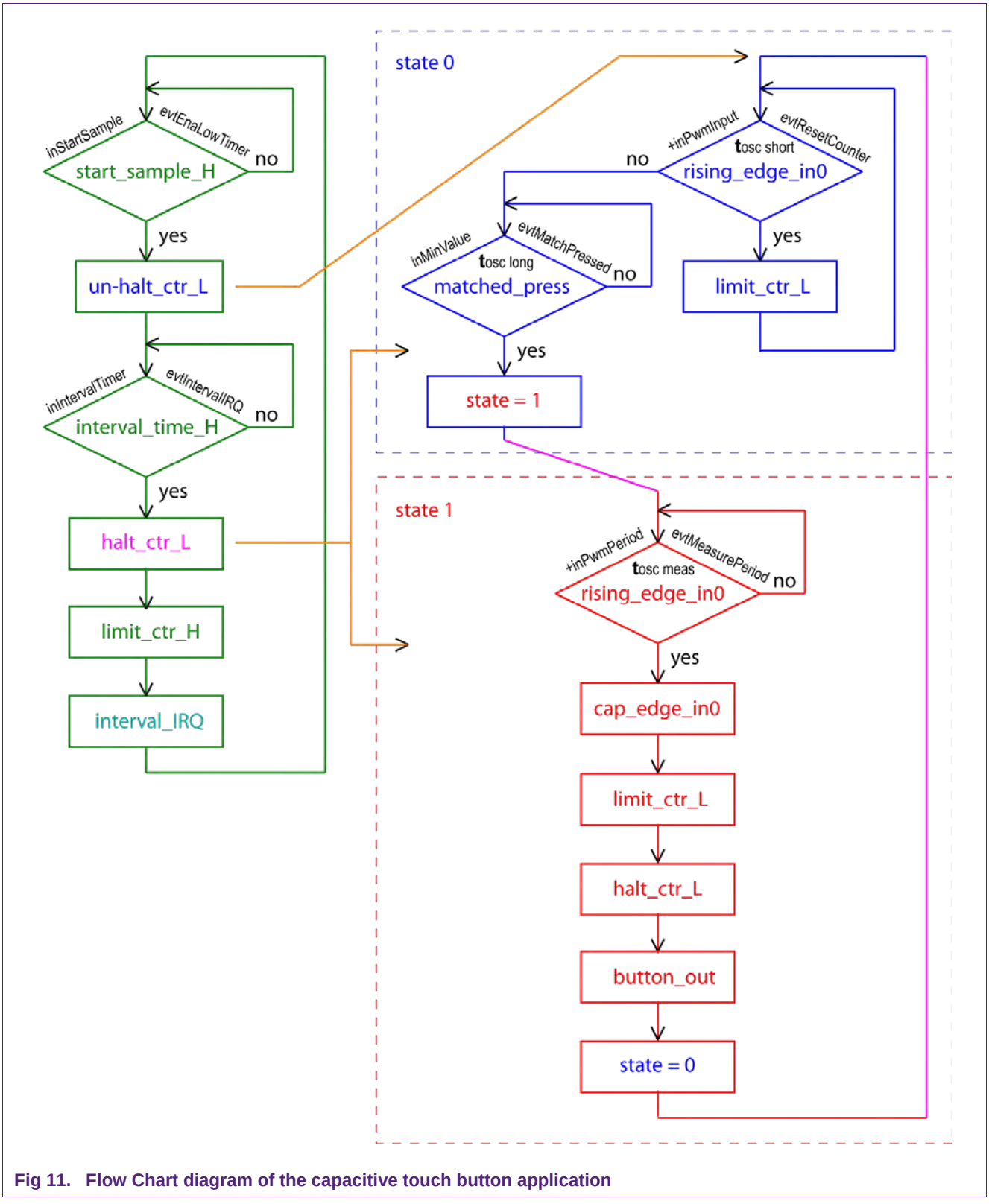


Fig 11. Flow Chart diagram of the capacitive touch button application

```
void sct_fsm_init (void)
{
LPC_SCT->CONFIG = (LPC_SCT->CONFIG & ~0x00060001) | 0x00000000; /* SPLIT */

/* MATCH/CAPTURE registers */
LPC_SCT->REGMODE_L = 0x0002;  /* L: 1x MATCH, 1x CAPTURE, 3 unused */
LPC_SCT->REGMODE_H = 0x0000;  /* H: 2x MATCH, 0x CAPTURE, 3 unused */
LPC_SCT->MATCH_H[0]    = eVal_IntervalTimer;    /* inIntervalTimer */
LPC_SCT->MATCHREL_H[0] = eVal_IntervalTimer;
LPC_SCT->MATCH_L[0]    = eVal_MinValue;         /* inMinValue */
LPC_SCT->MATCHREL_L[0] = eVal_MinValue;
LPC_SCT->MATCH_H[1]    = eVal_StartSample;      /* inStartSample */
LPC_SCT->MATCHREL_H[1] = eVal_StartSample;
LPC_SCT->CAPCTRL_L[1] = 0x0010;

/* OUTPUT registers */
LPC_SCT->OUT[1].SET = 0x00000010;        /* outButtonPress */
LPC_SCT->OUT[1].CLR = 0x00000000;
  /* Unused outputs must not be affected by any event */
LPC_SCT->OUT[0].SET = 0;
LPC_SCT->OUT[0].CLR = 0;
LPC_SCT->OUT[2].SET = 0;
LPC_SCT->OUT[2].CLR = 0;
LPC_SCT->OUT[3].SET = 0;
LPC_SCT->OUT[3].CLR = 0;
LPC_SCT->OUTPUT = (LPC_SCT->OUTPUT & ~0x00000000) | 0x00000002;

/* Conflict resolution register */

/* EVENT registers */
LPC_SCT->EVENT[0].CTRL = 0x00001010;     /* H: */
LPC_SCT->EVENT[0].STATE = 0xFFFFFFFF;
LPC_SCT->EVENT[1].CTRL = 0x00001011;     /* H: */
LPC_SCT->EVENT[1].STATE = 0xFFFFFFFF;
LPC_SCT->EVENT[2].CTRL = 0x00006400;     /* L: --> state L_ENTRY */
LPC_SCT->EVENT[2].STATE = 0x00000001;
LPC_SCT->EVENT[3].CTRL = 0x0000D000;     /* L: --> state state_1 */
LPC_SCT->EVENT[3].STATE = 0x00000001;
LPC_SCT->EVENT[4].CTRL = 0x00006400;     /* L: --> state L_ENTRY */
LPC_SCT->EVENT[4].STATE = 0x00000002;
  /* Unused events must not have any effect */
LPC_SCT->EVENT[5].STATE = 0;

/* STATE registers */
LPC_SCT->STATE_L = 0;
LPC_SCT->STATE_H = 0; /* implicit value */

/* state names assignment: */
  /* State L 0: L_ENTRY */
  /* State L 1: state_1 */

/* CORE registers */
LPC_SCT->START_L = 0x0002;
LPC_SCT->STOP_L =  0x0000;
LPC_SCT->HALT_L =  0x0011;
LPC_SCT->LIMIT_L = 0x0014;
LPC_SCT->START_H = 0x0000;
LPC_SCT->STOP_H =  0x0000;
LPC_SCT->HALT_H =  0x0000;
LPC_SCT->LIMIT_H = 0x0001;
LPC_SCT->EVEN =    0x00000001;
}
```

Fig 12. SCT Initialization Code function produced by Red State

Fig 13 shows how the main function calls the sct initialization function (sct_fsm_init). After the initialization, both the low and high counters are halted, so the code changes the status of the low counter to stopped, but not halted, while the high counter is unhalted and begins counting.

The SCTimer/PWM interrupt is enabled by NVIC_Enable(SCT_IRQn).

```
/* Enable AHB clock to the SCT. */
  LPC_SYSCTL->SYSAHBCLKCTRL |= (0x1 << 8) ;
  sct_fsm_init() ;

  LPC_SCT->EVFLAG = 0x3F ;          /* clear all SCT interrupt flags */
  NVIC_EnableIRQ(SCT_IRQn) ;     /* Enable SCT interrupt       */

  uint32_t tmp ;
  tmp  = LPC_SCT->CTRL_U ;
  tmp &= ~((1 << 2+16) |         /* un-halt counter_H */
           (1 << 2)) ;           /* un-halt counter_L */
  tmp |=   (1 << 1) ;            /*    stop counter_L */
  LPC_SCT->CTRL_U = tmp ;        /* The STOP and HALT bit need to be */
                                 /* updated in a single command      */
```

Fig 13. Starting the SCT timer

6. SCTimer/PWM interrupt

6.1 Interrupt processing

Button presses are detected when the period of the relaxation oscillator exceeds a specified value. The actual button press determination is made in the SCTimer/PWM interrupt. Every 2 ms the interrupt will occur and it will determine if a button was pressed, and what action should be taken. In the example code included with this application note, an LED is used to indicate when a button is pressed (level-sensitive), while the other LED toggles on the rising edge of a button press (edge-sensitive).

The interrupt service routine also sets up the comparator inputs to scan the next button. An oscilloscope trace displaying the scan change between the two buttons is shown in Fig 14.

Fig 14. Switching between the two buttons

6.2 Interrupt CPU bandwidth usage

The processor spends about 12 usec for each 2 msec time period in the IRQ which relates to a bandwidth usage of 0.6 % CPU time.

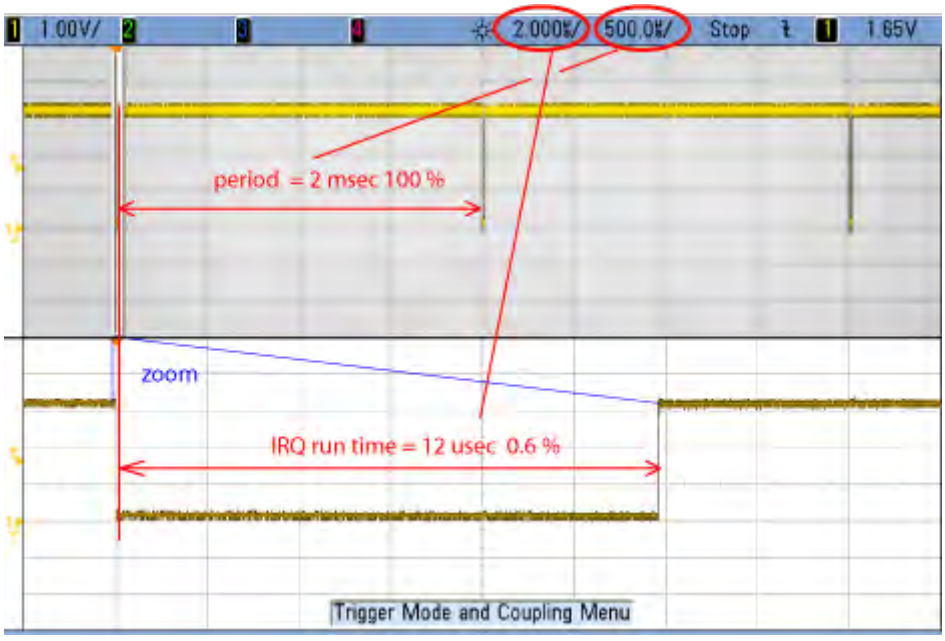


Fig 15. 0.6 % of CPU bandwidth spent in IRQ

7. LPCXpresso Red State tool explained

The LPCXpresso IDE provides a tool called Red State, a graphical state machine editor.

More details are available in the Code Red, Getting Started with Red State user manual. Since the State Machine design tool's GUI provides a very high level point of view, it does not directly correlate to the SCTimer/PWM control registers and the terms used by the SCTimer/PWM. This part of the document should assist to make the connections from the Red State design interface to SCTimer/PWM registers.

7.1 State diagram

The state diagram describes the global interactions of the individual states and events controlling the SCTimer/PWM.

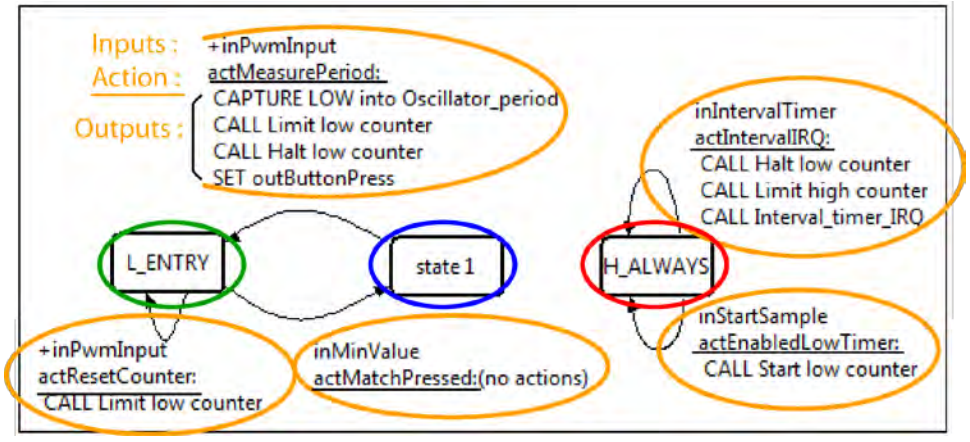


Fig 16. State diagram

7.1.1 States

The tool provides two kinds of states: regular states and pseudo states. Regular states are individual states mapped to the LPC_SCT->State_L, _H, _U registers, while the pseudo states are the accumulation of actions (SCTimer/PWM events) with their inputs and outputs which are assigned, and happen in, every (all) states.

Graphically adjacent to the states are the SCTimer/PWM events running in the particular state.

An SCTimer/PWM event can be assigned to multiple states.

H_ALWAYS: Pseudo state of Counter_H.

L_ENTRY: Entry, start state of Counter_L. State_L 0, the default start state upon reset.

State 1 : State 1 of Counter_L.

7.1.2 Actions (SCTimer/PWM events)

Actions (SCTimer/PWM events) are definitions of input stimuli, counter matches, output and/or state change actions.

A '+' or '–' sign indicates the input signal edge specified on an input pin trigger selection.

- The action 'actEnableLowTimer' is caused by state machine input, a Match_H register counter match and a state machine output will start Counter_L. Since this is connected with the pseudo state ALWAYS_H it will run in all Counter_L states.
- The action 'actMatchPressed' is caused by state machine Input, pressing a pad while in State_L 0 (L_ENTRY), lowering the oscillation frequency and therefore exceeding the timer (Counter_L) match value inMinValue (match event) before a rising edge (relaxation oscillator) on inPwmInput occurs which would Limit_L (reset) the counter.

  This action causes State_L 1 to be loaded.
- The action 'actMeasurePeriod' is caused by state machine input, a rising edge (relaxation oscillator) on inPwmInput while in State_L 1.

  This action causes the following Red State outputs: Capture Counter_L value, Limit Counter_L, Halt Counter_L, set Button_Pressed output and Load State_L 0 (L_ENTRY).

7.2 State table

The state table shows the actions and SCTimer/PWM events used by this state machine.

As mentioned earlier, H_ALWAYS are the Pseudo States running in all Counter_H States.

Action (SCTimer/PWM Event) actResetCounter is triggered by signal sigPulseStart (+edge on inPwmInput) and reloads the State_L (HEvent bit = 0) with the same value and does not cause a state change.

Action (SCTimer/PWM Event) actMatchPressed is triggered by signal sigButtonPress (match or exceed inMinValue) and loads the State_L (HEvent bit = 1) with State_L 1 … and so forth.

The signals column defines the signals causing the actions (SCTimer/PWM Events).

The Red State tool assigns the state table order arbitrarily during the editing process. The priority column will be necessary to assign a priority to states with the same trigger signal while in the same state to pick the highest as the one determining the state change being acted upon.

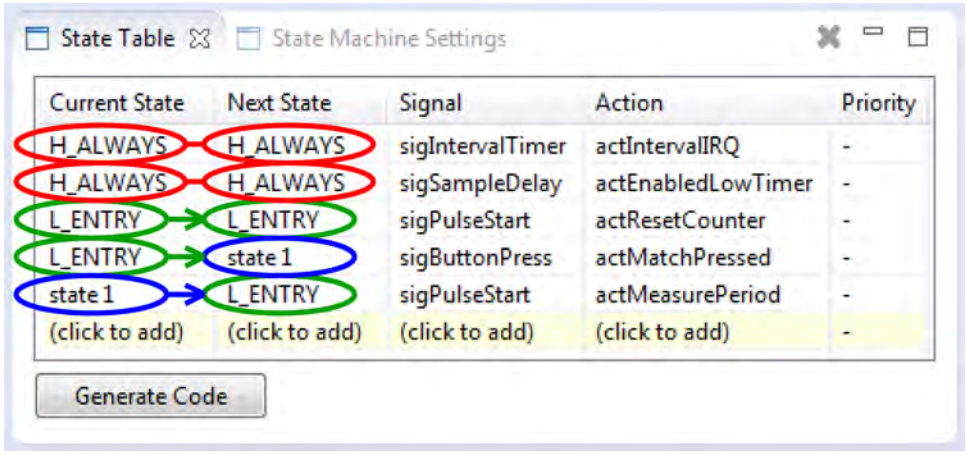
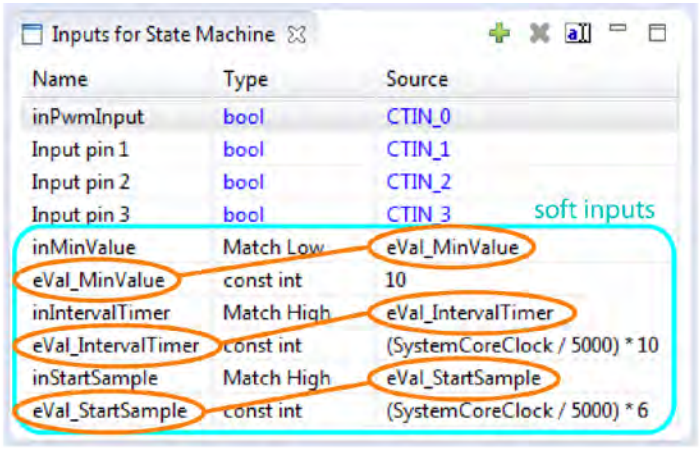


Fig 17. State table

7.3 State machine inputs

The state machine inputs table lists all available inputs, standard input signals and user defined soft inputs.

A “const int” can be seen as an enum constant used by match set values and similar. It also allows for expression evaluation like “(SystemCoreClock / 5000) * 10”.




```
LPC_SCT->MATCH_H    [0] = eVal_IntervalTimer ;      /* inIntervalTimer */
LPC_SCT->MATCHREL_H[0] = eVal_IntervalTimer ;
LPC_SCT->MATCH_L    [0] = eVal_MinValue ;           /* inMinValue      */
LPC_SCT->MATCHREL_L[0] = eVal_MinValue ;
LPC_SCT->MATCH_H    [1] = eVal_StartSample ;        /* inStartSample   */
LPC_SCT->MATCHREL_H[1] = eVal_StartSample ;
```

Fig 18. State machine inputs

Also indicated is the affected part of the code generated.

7.4 State machine outputs

The state machine outputs table lists all available outputs, standard output signals and user definable register access.

This list also provides init values for the SCTimer/PWM output registers by use of the preload column.

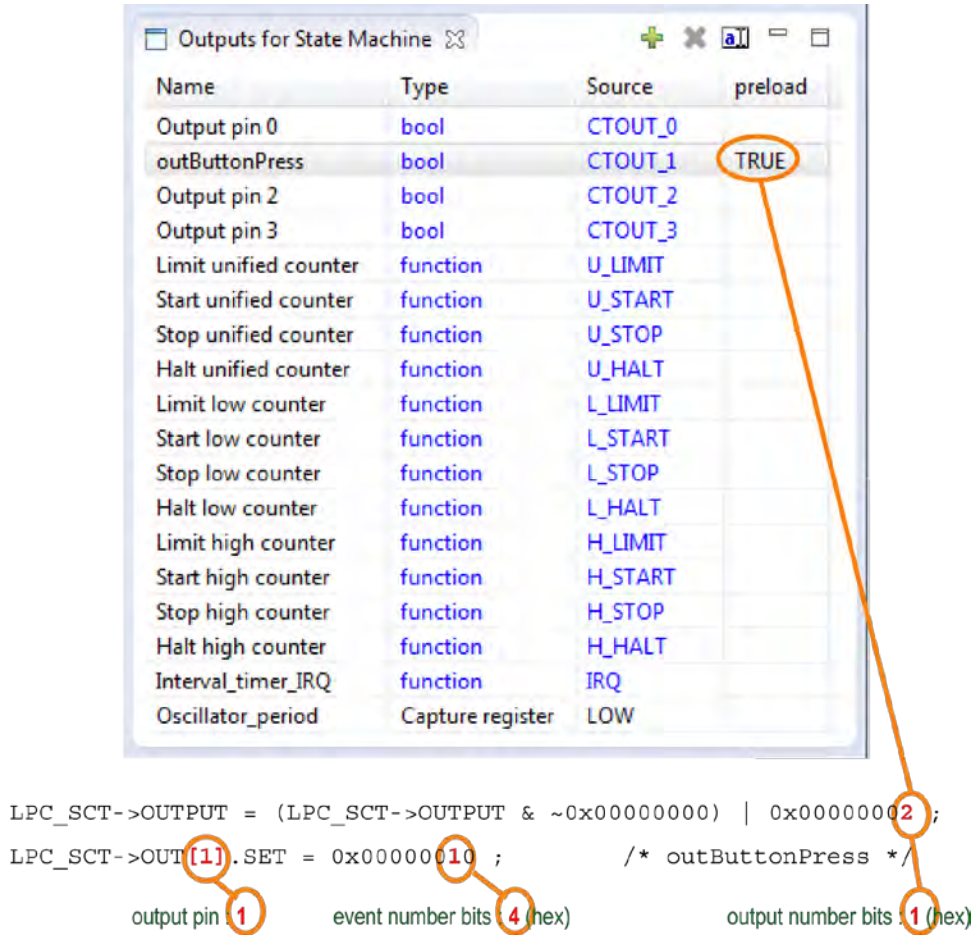


Fig 19. State machine outputs

Also indicated is the affected part of the code generated along with reference points and lines to its relevant data, bit positions and array indices.

7.5 State machine action list

The state machine action list defines the output operations performed by each action (SCTimer/PWM Event).

An action (SCTimer/PWM Event) item with operation “none” assigned might still have a hidden operation assigned, like a state load or add operation which are only indicated in the state diagram and the state table.

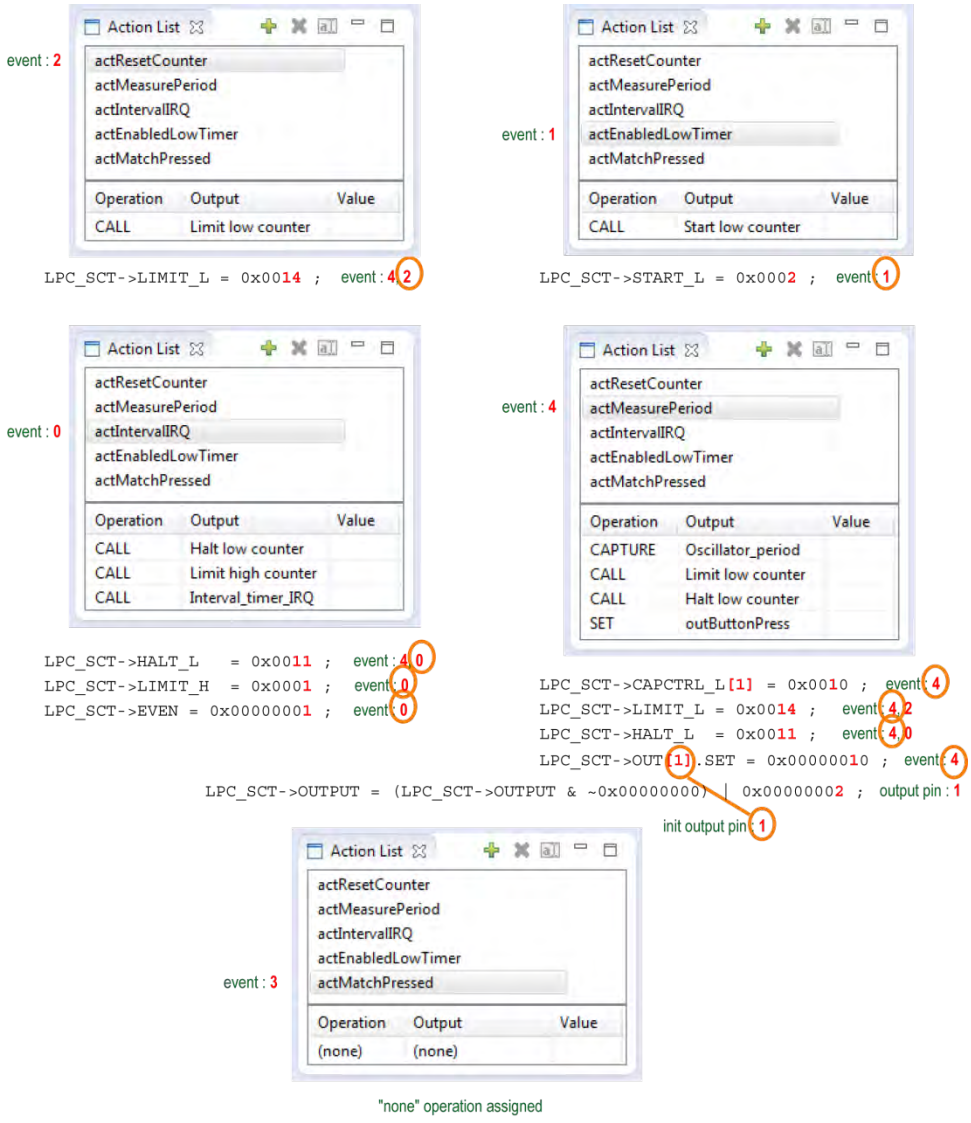


Fig 20. State machine action list

Also indicated is the affected part of the code generated along with reference points and lines to its relevant data, bit positions and array indices.

Action (SCTimer/PWM event) actMatchPressed does not show an associated output operation, but has the state change from State_L 0 to State_L 1 shown in the state diagram and state table.

7.6 State machine signals

The state machine signals define the event trigger signals.

The included code snippets and lines to the event register bits help associating individual event register bits to all related other register data entries and register array indices.

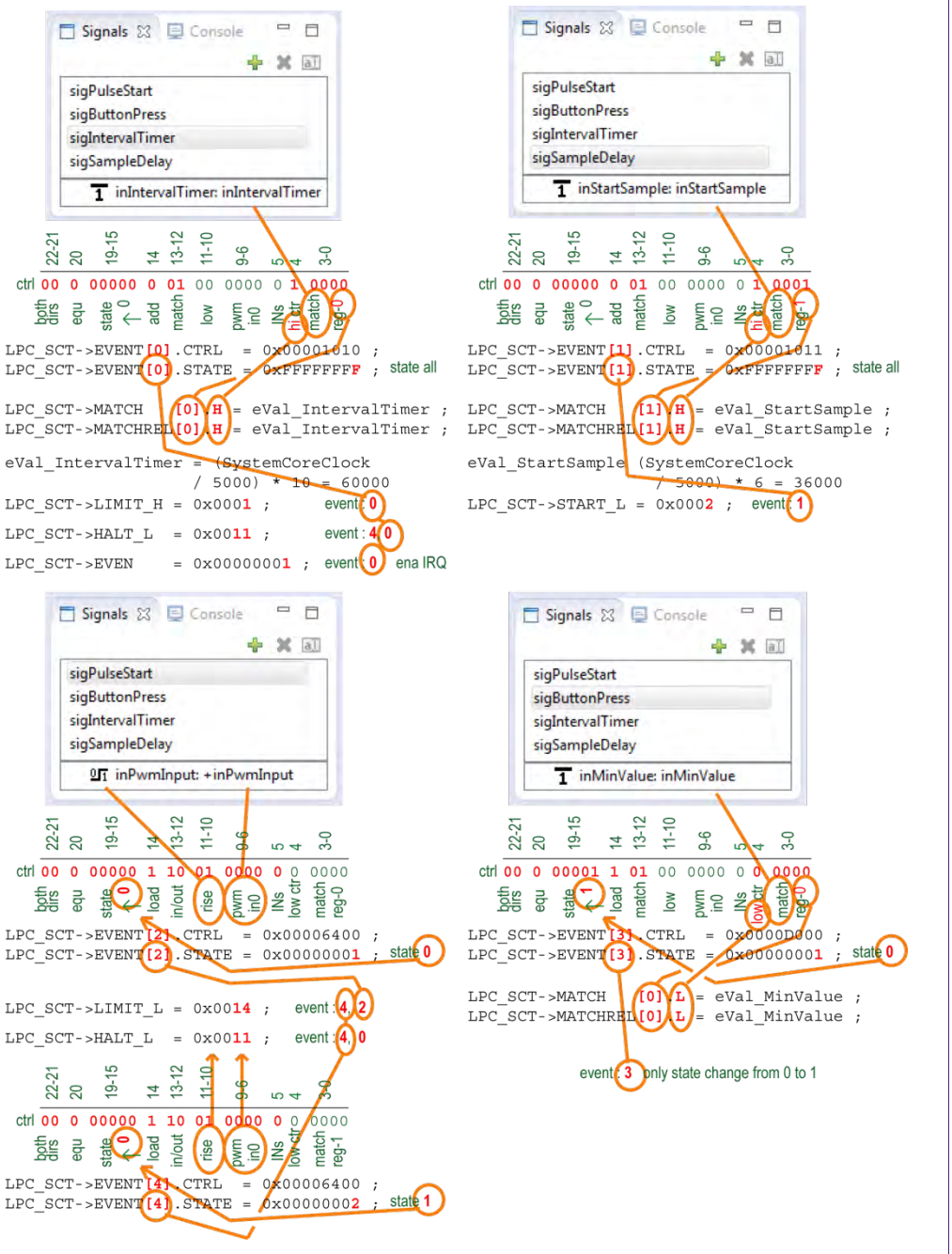


Fig 21. State machine signals

Signal sigIntervalTimer and sigStartSample provide the event trigger definitions for event 0 and event 1 being a Match_H 0 and Match_H 1 condition.

These two Counter_H related events run on all states and provide the following operations for event 0, Limit_H, Halt_L, fire IRQ and for event 1 the following operation, Start_L.

The state is not getting changed since 0 is being added to the current state.

Signal sigPulseStart provides the event trigger of a rising edge for event 2 and event 4. Event 2 has the following operation associated, Limit_L while event 4 associates operation Limit_L and Halt_L and changes from state 1 to 0.

Signal sigButtonPress triggers event 3 upon a Match_L 0 condition. The operations assigned to it are “none”, other than a state change from 0 to 1.

8. SCTimer/PWM const symbols and macros

Besides using the Red State tool, another approach for generating the SCTimer/PWM register initialization content would be to use a large set of enum and #define register and register-bit definitions along with some macros to provide an easier to oversee SCTimer/PWM content of registers along with its application flow.

Three files have been included with this app-note to illustrate the use of predefined constants.

1. sct_defs.h provides generic SCTimer/PWM register and register-bit definitions.
2. sct_ctouch.h contains capacitive touch app specific SCTimer/PWM constant definitions and macros.
3. sct ctouch.c holds the SCTimer/PWM initialization code utilizing the above mentioned include files and their symbolic definitions.

9. SCTimer/PWM register content

A spreadsheet example can be used to illustrate where the SCTimer/PWM data is being placed.

It serves also as a nice configuration tool by filling in the desired options and at the end calculating the resulting register content.

Since the whole LPC8xx SCTimer/PWM spreadsheet size exceeds a regular page size, only a small portion is included as a sample.

The spreadsheet-file should be part of this appnote documentation.

		EVENT STATE REGs [5]					EVENT CTRL REGs [6]										
					match = 4												
					match = 3		io = 3										
					match = 2	AND = 3	io = 2	high = 3									
					match = 1	I / O = 2	io = 1	fall = 2								down = 2	
				high ctr = 1	match = 0	match = 1	io = 0	rise = 1	out = 1	load = 1	state = 1	range = 1	up = 1				
		x	x	low ctr = 0	reg name	OR = 0	io name	low = 0	in = 0	add = 0	state = 0	equ = 0	bi = 0				
		state 0	state 1	ctrl 4	ctrl 3 – 0	ctrl 13 – 12	ctrl 9 – 6	ctrl 11 – 10	ctrl 5	ctrl 14	ctrl 19 – 15	ctrl 20	ctrl 22 – 21	31 – 23			
name str	event	state 0	state 1	hevent	matchsel	combmode	iosel	iocond	outsel	stateId	statev	matchmem	direction	space			
evtIntervalTimer	0	1	1	1	regIntervalTimer	match											
evtStartSample	1	1	1	1	regStartTimer	match											
evtPwmInput	2	1	0	0		I / O	inPwmInput	rise	in	load	staEntry						
evtMinValue	3	1	0	0	regStartTimer	match				load	staState1						
evtMatchPressed	4	0	1	0		I / O	inPwmInput	rise	in	load	staEntry						
event 5	5																

		OUT SET / CLR REGs [4]							
		x	x	x	x	x	x	x	x
		outset 0	outclr 0	outset 1	outclr 1	outset 2	outclr 2	outset 3	outclr 3
name str	event	outset 0	outclr 0	outset 1	outclr 1	outset 2	outclr 2	outset 3	outclr 3
evtIntervalTimer	0								
evtStartSample	1								
evtPwmInput	2								
evtMinValue	3								
evtMatchPressed	4			1					
event 5	5								
				0x10					

		CAPCON REGS [5]									
		x	x	x	x	x	x	x	x	x	x
		capcon 0 L	capcon 0 H	capcon 1 L	capcon 1 H	capcon 2 L	capcon 2 H	capcon 3 L	capcon 3 H	capcon 4 L	capcon 4 H
name str	event	capcon 0 L	capcon 0 H	capcon 1 L	capcon1 H	capcon 2 L	capcon 2 H	capcon 3 L	capcon 3 H	capcon 4 L	capcon 4 H
evtIntervalTimer	0										
evtStartSample	1										
evtPwmInput	2										
evtMinValue	3										
evtMatchPressed	4			1							
event 5	5										
				0x10							

Fig 22. SCTimer/PWM register content

10. Legal information

10.1 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.

10.2 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. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

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 and its suppliers accept 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.

10.3 Trademarks

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

11. List of figures

12. Contents

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

Date of release: 14 April 2014
Document identifier: AN11536