TN1222
Technical note

How to inject “Fake Fault” in the FCCU

Introduction

SPC570Sxx devices embed multiple modules to support the functional safety (please check Reference Manual of SC57xx and SPC58xx to verify if this TN is applicable see *Section B.1: Reference document*).

Fault Collection Control Unit (FCCU) is one of these. It collects faults coming from all hardware monitors embedded in the device.

In case of fault, it can move the device into the safe state without any CPU intervention.

FCCU and the whole error reaction path are prone to latent failures that are detected by the application at least once per trip time. Some measures to detect latent failures run typically once after the boot before starting the safety application.

To satisfy such requirement, the FCCU is protected by 2 hardware mechanisms:

1. LBIST and
2. FOSU that monitors the FCCU reaction.

These measures cannot detect failures affecting the whole reaction whose integrity is verified by additional mechanisms.

The FCCU includes a feature called “Fake Fault” injection that allows injecting a fault and verifying the integrity of the whole reaction path.

This document explains how to inject a “Fake Fault” and the right procedure to clear it.

An example on SPC570Sxx has been developed. This example can be extended to all devices which implement the FCCU.

As prerequisite, the reader should have a basic knowledge on how to configure a FCCU module.

Contents

List of tables

List of figures

1 Overview

The Functional Safety chapter of the RM of SPC570Sxx (see *Section B.1: Reference document*) microcontroller describes the safety features of the device.

The hardware redundancy and specific IPs modules have been designed to increase the level of Safety of the system/ECU[b].

The main IPs designed for this task are:

- STCU: to run L/MBIST in order to test analog parts and RAM memories
- MEMU: to collect and report errors coming from system memories including both volatile and non-volatile ones
- FCCU: to collect and detect latent faults coming from monitors modules (such as CMU, RCCU,LVD,HVD and so on…)

FCCU integrity is guaranteed by:

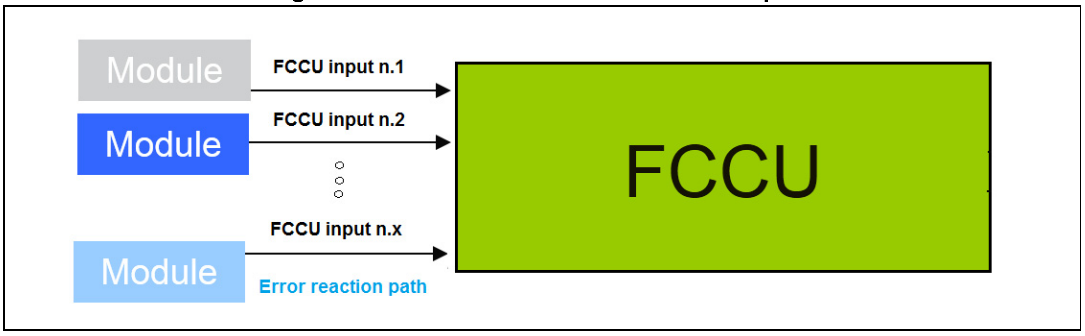
- FOSU that monitors whether the FCCU reacts to a fault within the expected time.
- LBIST that includes the FCCU
- Injecting a “Fake Fault” in the FCCU to verify the integrity of the whole reaction path.

First two mechanisms are pure hardware based. The third one requires some software.

The list of the FCCU failure inputs is provided in the Functional Safety chapter of RM see *Section B.1: Reference document*. Both kinds of fault (NCF or CF[c]) can be injected as “Fake Fault”.

Figure 1 shows the connection between the failure inputs and the FCCU:

Figure 1. FCCU connected with Fault Inputs



The FCCU is based on a state machine (FSM) that implements 4 different states:

- <u>NORMAL</u>: it’s the default state if no fault is detected
- <u>CONFIG</u>: software moves the FSM from NORMAL to CONFIG state to configure the FCCU.

b. In this way the device is kept under the acceptable risk threshold according with ISO 262622 standard.

c. It depends on the reference document, Non critical faults (NCF) are also known as Recoverable faults (RF) and Critical faults (CF) are also known as Unrecoverable faults (UF)

A configurable hardware timeout monitors how long the FSM remains in CONFIG state. If this time is too long, the hardware moves the FSM back to the NORMAL state and the previous configuration is maintained.

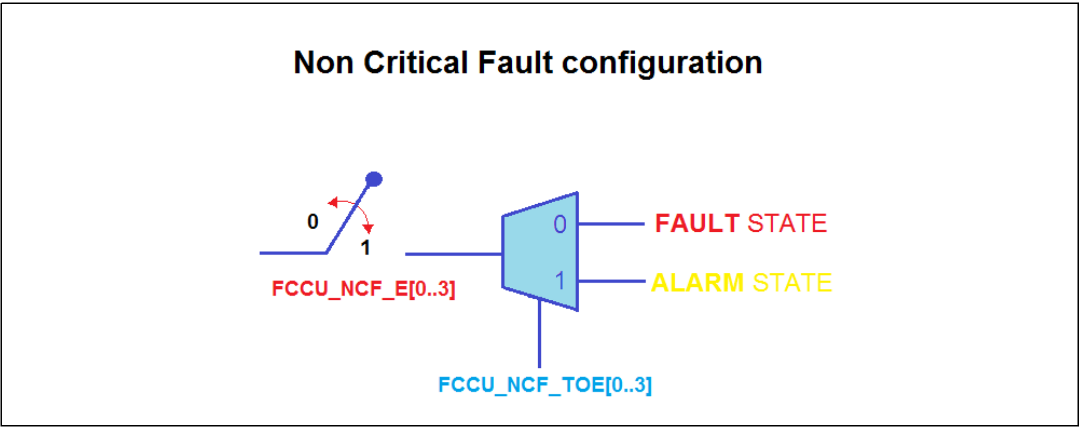
- ALARM: the FSM moves to ALARM state in case a fault is detected. In this state the software can take some preliminary reaction to the fault. If the source of the failure is not cleared within a configurable timeout the FSM goes to Fault state and the FCCU moves the MCU to Safe state.
- FAULT: In this state, the FCCU moves the MCU to Safe State[d] by taking some configurable reaction (e.g. Functional Reset and/or Error out).

Transition to ALARM state can be disabled by software. In this case the FCCU state machine transits directly to the Fault state.

Depending on the setting of 1) NCF_Ex (to enable or not the specific Non Critical Fault (RF)"x") and 2) NCF_TOEx registers, the FSM moves in ALARM state or not (see ALARM and FAULT configuration):

- FSM to FAULT STATE: NCF_Ex =1 and NCF_TOEx = 0
- FSM to ALARM STATE: NCF_Ex =1 and NCF_TOEx = 1.

Figure 2. ALARM and FAULT configuration



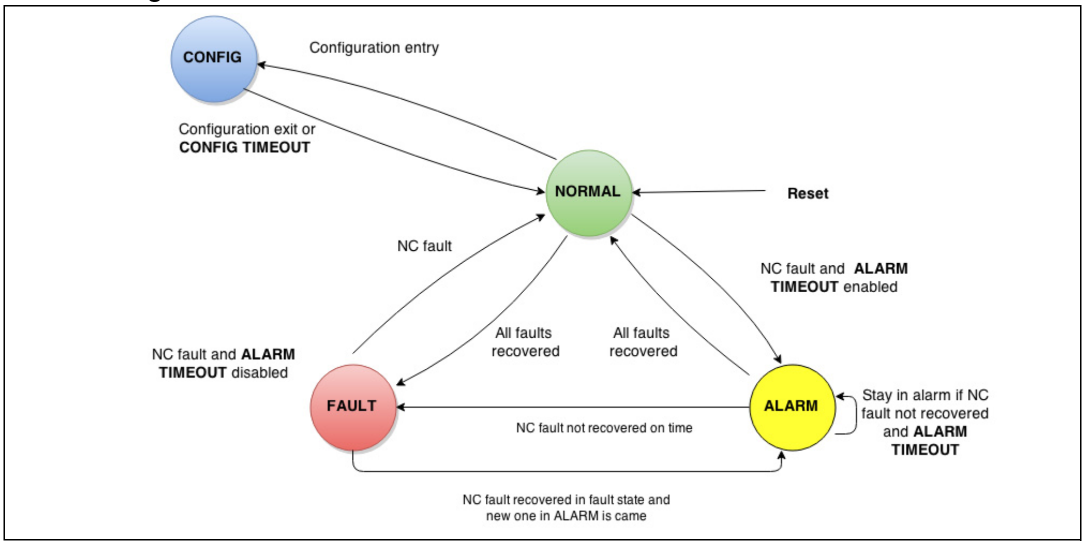
If ALARM state is enabled, NCF_TO register defines the timeout for the recovery of the non-critical faults (i.e.ALARM timeout).

Once FCCU enters in ALARM state, following the assertion of a non-critical fault enabled (NCF_Ex and NCF_TOEx are set), the timer starts the count down. If the time expires, and the fault has not been cleared yet, the FSM moves to FAULT state.

Figure 3 describes the 4 states of the FSM of the FCCU with timeout associated:

d. Assumed safe states are listed in the Safety Manual of the device under analysis (see *Section B.1: Reference document*).

Figure 3. The 4 states of FSM of the FCCU with timeout associated


CONFIG
Configuration entry
Configuration exit or CONFIG TIMEOUT
NORMAL
Reset
NC fault
NC fault and ALARM TIMEOUT enabled
All faults recovered
All faults recovered
NC fault and ALARM TIMEOUT disabled
FAULT
ALARM
Stay in alarm if NC fault not recovered and ALARM TIMEOUT
NC fault not recovered on time
NC fault recovered in fault state and new one in ALARM is came

The integrity of the FCCU is also monitored by its supervisor, called FOSU. It contains a timeout with a fixed duration (i.e. FOSU_COUNT driven by the IRCOSC, 16 Mhz).

FOSU monitors whether the FCCU reacts within the FOSU timeout. Expected FCCU reaction is one of the following actions:

- Reset (short or long)
- IRQ
- NMI
- Interrupter
- Err out triggered[e]

If the FOSU does not observe one of the above FCCU reaction within FOSU timeout, it assumes that the FCCU is faulty[f] and it triggers a destructive reset.

Since the FCCU does not react to a fault while its state machine is in CONFIG state, it should not be kept in CONFIG state for longer than the FOSU_COUNT duration. Otherwise, there is a risk that an incoming error causes the FOSU seeing the FCCU as faulty and then resets the MCU.

When a fault happens, FOSU timer starts together with the ALARM timer (if the FCCU is configured to go to ALARM state).

e. Failure of the MCU is signaled via two pins: EOUT0 and EOUT1. There are different configurable protocols to indicate a failure via the one or two pins. See RM for further details (see *Section B.1: Reference document*

f. FCCU has not reacted because its functionality is impaired by a latent fault. In this case there is the accumulation of 2 faults. A first fault detected by a monitor and forwarded to the FCCU. A second fault, potentially latent, is affecting the FCCU.

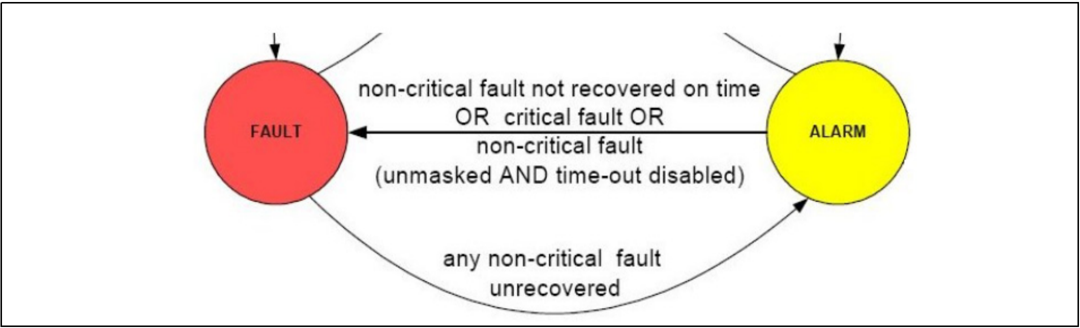
While FOSU timer is running, any subsequent captured fault neither restart nor reinitialize the FOSU timer. The FOSU timer stops when the FCCU shows any of the possible reactions:

- Reset (short or long)
- IRQ
- NMI
- Interrupt
- Error out triggered

Therefore, 3 different timeout clocked by the IRCOSC are defined in the FCCU:

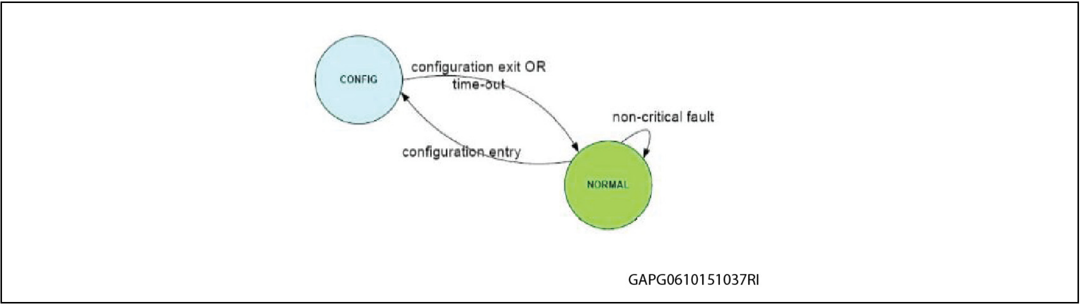
- **Alarm Timeout**: when in alarm state, if fault is not cleared within this timeout (FCCU_NCF_TO register), the FSM goes to FAULT state.

Figure 4. FAULT and ALARM state transition


FAULT
ALARM
non-critical fault not recovered on time OR critical fault OR non-critical fault (unmasked AND time-out disabled)
any non-critical fault unrecovered

- **Config Timeout**: it is a sort of watchdog timer. If FCCU stays in CONFIG state longer than timeout (FCCU_CFG_TO), the FSM goes automatically back to NORMAL state and the previous configuration of the FCCU is restored.

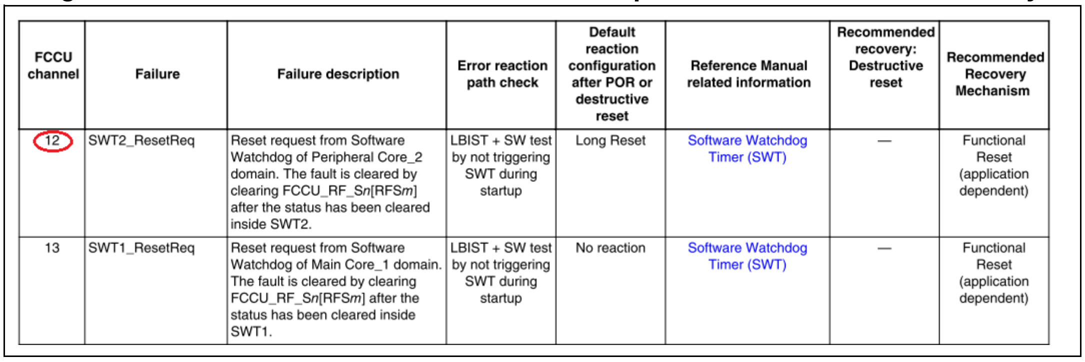
Figure 5. CONFIG state transition



- **FOSU timeout**: if FCCU does not react to an incoming failure within this timeout a destructive reset happens.

2 “Fake Fault” injection

Figure 6. Extract of a section of FCCU failure inputs of RM of SPC570Sxx family



FCCU channel	Failure	Failure description	Error reaction path check	Default reaction configuration after POR or destructive reset	Reference Manual related information	Recommended recovery: Destructive reset	Recommended Recovery Mechanism
12	SWT2_ResetReq	Reset request from Software Watchdog of Peripheral Core_2 domain. The fault is cleared by clearing FCCU_RF_S*n*[RFS*m*] after the status has been cleared inside SWT2.	LBIST + SW test by not triggering SWT during startup	Long Reset	Software Watchdog Timer (SWT)	—	Functional Reset (application dependent)
13	SWT1_ResetReq	Reset request from Software Watchdog of Main Core_1 domain. The fault is cleared by clearing FCCU_RF_S*n*[RFS*m*] after the status has been cleared inside SWT1.	LBIST + SW test by not triggering SWT during startup	No reaction	Software Watchdog Timer (SWT)	—	Functional Reset (application dependent)

Depending on how the FCCU reaction is configured for this fault, there are two possibilities to clear the fault and move the state machine back to NORMAL state:

1. clearing NCF_S12 when FCCU is in FAULT state, or
2. clearing NCF_S12 when FSM is in ALARM state[g]. This action is done before ALARM timeout, otherwise the state machine move to the FAULT state.

The second option is implemented by the example described in this document (see *Section Appendix A: Code to inject a fault*).

2.1 Fault injection with ALARM state enabled

Since the FOSU waits for a reaction by FCCU, clearing the FCCU_NCF_Sx, it may not be enough to go back to the NORMAL state and to continue the execution of the application.

FOSU expects one of the following FCCU reactions within a FOSU timeout[h]:

- Reset: long or short functional reset
- IRQ: NMI or Alarm
- Error out triggered (by FCCU or by SW)

The example here proposed has the ALARM state activated on fault #12. The comeback to NORMAL state is possible, if the following two actions hereafter are satisfied:

- Enabling IRQ alarm on fault error source x (IRQ_ALARM_EN0[12])
- Clearing NCF_S12 when FSM is in ALARM state

Enabling IRQ in alarm state is the reaction that stops the FOSU timeout. Observing this reaction the FOSU verifies the integrity of the FCCU which has reacted, with an interrupt, within the expected time.

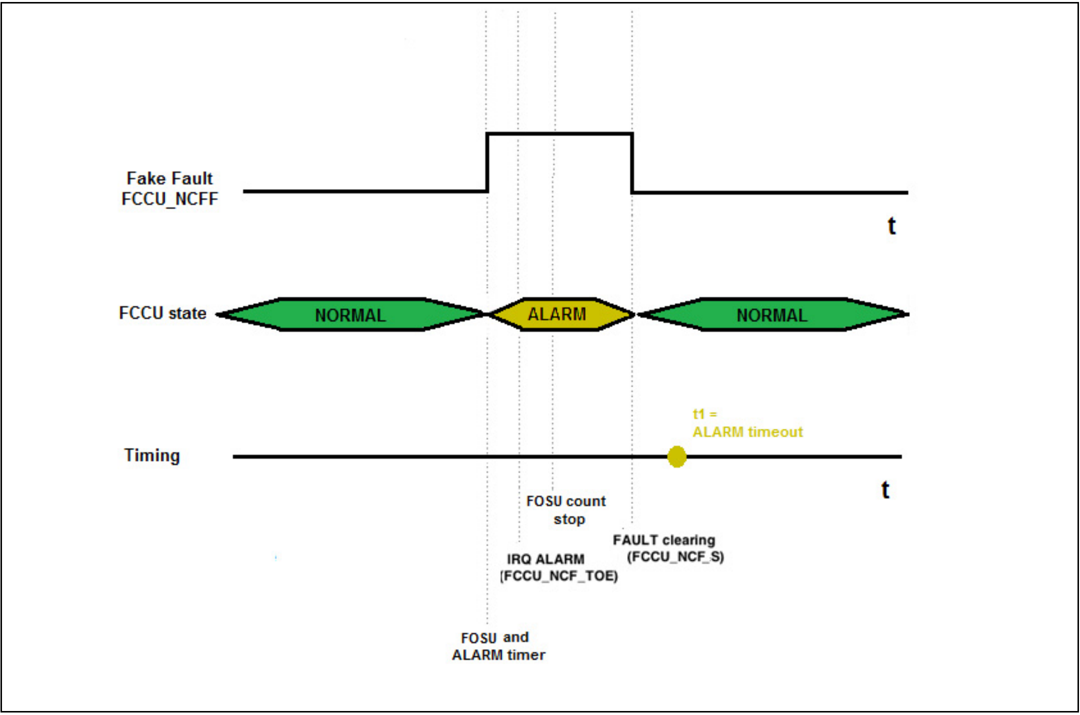
If the IRQ alarm is not enabled, the software can clear the fault and FCCU state machine goes back to NORMAL state. But since the FOSU has not observed any FCCU reaction. As soon as the FOSU timeout expires a destructive reset is triggered by the FOSU.

g. This means that ALARM timer is activated via NCF_TO register.

h. Otherwise it triggers a destructive reset.

Figure 7 and *Figure 8* show the different behavior when the interrupt on ALARM timeout is enabled or not.

Figure 7. Correct procedure of fault clearing in ALARM state

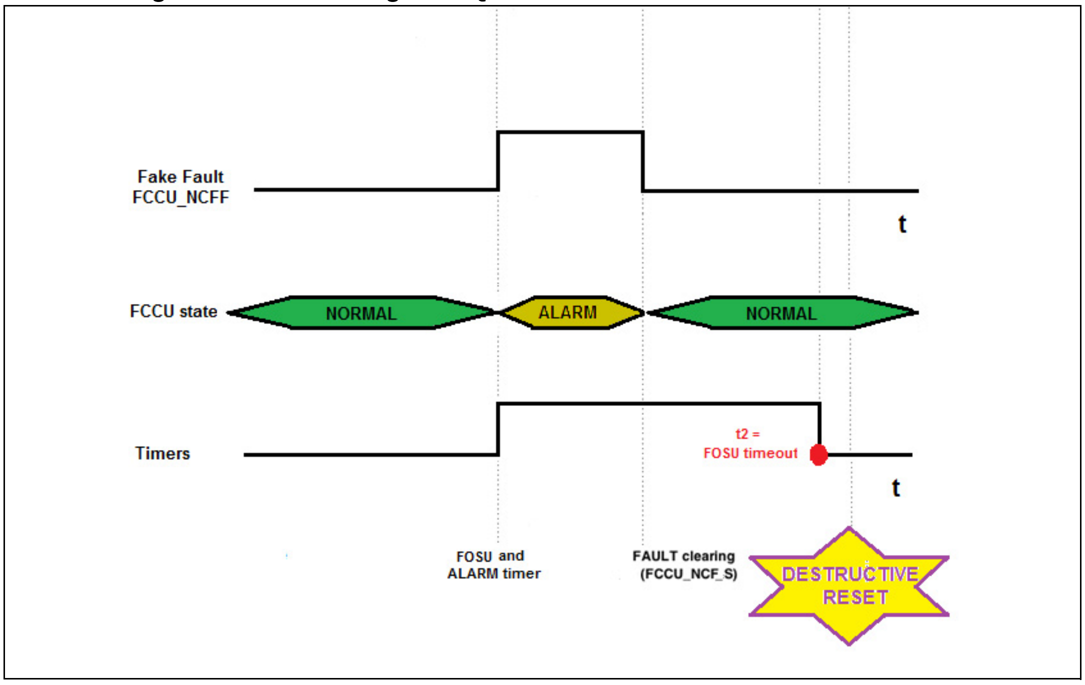


With reference to *Figure 7*, FCCU is sensitive to the edge of a signal fault, in this case the "Fake Fault" injected. Then FSM moves to ALARM state (label FOSU and ALMAN timer). FOSU and ALARM timer starts counting.

The IRQ ALARM is triggered after a couple of clock cycles. FOSU timer stops as the IRQ alarm is set. After that NCF is cleared, the FSM goes back to normal state (label FAULT clearing). *Figure 7* shows also the hypothetic ALARM timeout (*label t1=ALARM timeout*) which is not reached in this case.

A different scenario is shown in *Figure 8*, i.e. IRQ_ALARM is not activated. In this case, since no interrupt or other reaction is taken by the FCCU, FOSU timeout expires and a destructive reset is generated by the FOSU (although the FSM goes back in NORMAL STATE within the ALARM timeout).

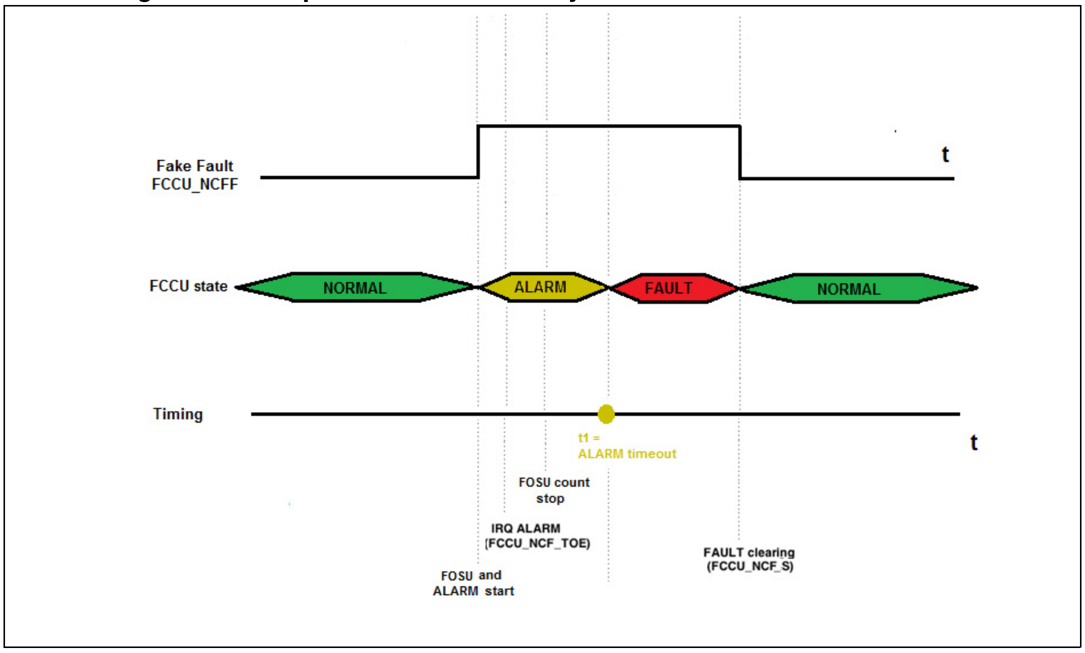
Figure 8. The missing of IRQ ALARM causes the destructive reset



Last proposed scenario appears if the fault is not cleared within *t1: ALARM timeout* (see *Figure 9*). In this case the FCCU state machine goes to FAULT state.

As soon as the software clears the fault, the FSM goes back in NORMAL state.

Figure 9. Example of "Fake Fault" injection and clear in FAULT state



2.2 How to configure FCCU

After reset, FCCU is in NORMAL state. Before trying to inject any fault it is necessary to configure the FCCU to enable the reaction to the fault that has to be activated. Let’s suppose to activate the fault #12.

The configuration consists of the following steps:

- Unlock FCCU (writing the key ->0x0000_00BC in the FCCU_TRANS_LOCK register)
- Clearing the NCF status registers ->FCCU_NCFSx
- Config timeout setting before entering in CONFIG state
- Moving versus CONFIG state
- In CONFIG state is configured the recoverable fault.
- In the example reported in *Section Appendix A: Code to inject a fault*), for reset reaction, it has been configured for all faults (FCCU.NCFS_CFG[z] -> 0x0)
- NCF #12 is enabled (FCCU_NCF_Ex)
- NCF #12 moves into the ALARM state is enabled (FCCU.NCF_TOEx)
- ALARM timeout setting (FCCU.NCF_TO)
- Interrupt enabling for NCF #12 (FCCU.IRQ_ALARM_ENx)
- Come back in NORMAL state

3 Summary

This document describes the procedure how to inject “Fake Fault” in the SPC570S device.

Before triggering the “Fake Fault”, it is necessary to configure the FCCU in order to enable a reaction to stop the FOSU counter.

The user has to pay attention in case the injected fault is cleared while the FCCU state machine is in ALARM state. In this case the Alarm IRQ is enabled. Otherwise the FOSU timer keeps counting even if the fault has been cleared. As result, FOSU triggers a destructive reset as soon as its timer expires.

In *Section Appendix A* there is an example code to inject fault n.12

Appendix A Code to inject a fault

In this section it is provided the code necessary to inject a fault in FCCU. The example is referred to fault 12, ma the concept can be extended to all failure inputs.

```
#include "..\platform\SPC570S50L.h"
#include "stdio.h"
#include "..\platform\spc57_init.h"


/*****************************************************************
| variables and typedef
|----------------------------------------------------------------*/

typedef enum {
   eFccu_NORMAL  = 0,
   eFccu_CONFIG = 1,
   eFccu_ALARM= 2,
   eFccu_FAULT= 3
} tenFccuState;

typedef struct {
   vuint32_t STAT;
   volatile tenFccuState FSM;
   vuint32_t RF_S0;
   vuint32_t RF_S1;
   vuint32_t RF_S2;
   vuint32_t RF_S3;
} tuFCCU_StatusLog;




extern void DELAY(tU32 u32delay);
extern void OUTPAD_Init(tU32 u32Pcr);
extern void OUTPAD_Toggle(tU32 u32Pcr);

void FCCU_Init(void);
void FCCU_error_injection(void);
void FCCU_CLEAR_FLAGS(void);
void FCCU_STATUS_CHK(vint32_t State_exp);
void FCCU_TransToConfig(void);
void FCCU_TransToNormal(void);
void FCCU_CONFIG_RF(void);
void FCCU_StatusLog(tuFCCU_StatusLog *FCCU_Log);


volatile uint32_t u32DebugFlag = 0;

tuFCCU_StatusLog udFCCU_Status_BeforeInjection;
tuFCCU_StatusLog udFCCU_Status_AfterInjection;
tuFCCU_StatusLog udFCCU_Status_AfterClearInj;
tuFCCU_StatusLog udFCCU_Status_AfterNormalInj;


tuFCCU_StatusLog udFCCU_Status_BeforeClear;
tuFCCU_StatusLog udFCCU_Status_AfterClear;
tuFCCU_StatusLog udFCCU_Status_AfterMovingToConfig;
tuFCCU_StatusLog udFCCU_Status_BackToNormal;

/******************************************************************
| defines and macros (scope: module-local)
|-----------------------------------------------------------------*/
#define PAD_LED_CORE20/* PA[0] */


#define TRANS_ABORT (vint32_t) 0x2
#define TRANS_SUCCESSFUL (vint32_t) 0x3


/* FCCU Status */
#define FCCU_STATE_NORMAL (vint32_t) 0x0
#define FCCU_STATE_CONFIG (vint32_t) 0x1
#define FCCU_STATE_ALARM  (vint32_t) 0x2
#define FCCU_STATE_FAULT  (vint32_t) 0x3

/* FCCU Operation run */
#define FCCU_OP1 (vint32_t) 0x1
#define FCCU_OP2 (vint32_t) 0x2
#define FCCU_OP3 (vint32_t) 0x3
#define FCCU_OP10 (vint32_t) 0xA




#define FCCU_RF_SWT_2 12

void main(void){
    tU32 u32Counter = 0;

    while(u32DebugFlag!=0xCCAA) { asm("nop"); };

    MCU_Init(); /* MODE_ENTRY initialization */

    OUTPAD_Init(44);  /* Configure GPIO as outputs */

    FCCU_Init();

    FCCU_error_injection();

    /* Main While Loop */

    while(1){
       DELAY(40000000);
       OUTPAD_Toggle(44);
       u32Counter++;
    }
}


void FCCU_Init()
{

    /*1. FCCU_TRANS_UNLOCK */
    FCCU.TRANS_LOCK.R = 0x000000BC;

    /* 2. clear of the status flags */
    FCCU_StatusLog(&udFCCU_Status_BeforeClear);
    FCCU_CLEAR_FLAGS();
    FCCU_StatusLog(&udFCCU_Status_AfterClear);

    /* 3.FCCU Time out Configuration */
    FCCU.CFG_TO.R= 0x00000007;

    /* 4.  FCCU moves in CONFIG state */
    FCCU_TransToConfig();

    FCCU_StatusLog(&udFCCU_Status_AfterMovingToConfig);

    /*5. Recoverable fault Configuration  - use ONLY in CONFIG state */
    FCCU_CONFIG_RF();

    /* 6. FCCU moves in NORMAL state */
```

```
   FCCU_TransToNormal();
   FCCU_StatusLog(&udFCCU_Status_BackToNormal);


}

void FCCU_error_injection()
{

/*fake fault injection */

   FCCU_StatusLog(&udFCCU_Status_BeforeInjection);

   FCCU.NCFF.R = FCCU_RF_SWT_2;

   FCCU_StatusLog(&udFCCU_Status_AfterInjection);

   DELAY(10); //a brief waiting time before clearing

   /* Clear of status flags */
   FCCU.NCFK.R = 0xAB3498FE;
   FCCU.NCFS[0].R = (1 << FCCU_RF_SWT_2);   //FCCU RF status register
   while(FCCU.CTRL.B.OPS != TRANS_SUCCESSFUL) { asm("nop"); };

   FCCU_StatusLog(&udFCCU_Status_AfterClearInj);

   DELAY(200);

   FCCU_StatusLog(&udFCCU_Status_AfterNormalInj);

}

void FCCU_CLEAR_FLAGS(void)
{

   /* clear of the SSCM.STATUS.CER flag (PER.S ANC:0xFFFF8000 %LONG
0xffffffff) */
   SSCM.STATUS.R = 0xFFFFFFFF;


/* Clear of status flags */
   FCCU.NCFK.R = 0xAB3498FE;
   FCCU.NCFS[0].R = 0xFFFFFFFF;   //FCCU RF status register
   /* wait for the completion of the CLEAR FLAGS operation */
   while(FCCU.CTRL.B.OPS != TRANS_SUCCESSFUL) { asm("nop"); };
```

```
    FCCU.NCFK.R = 0xAB3498FE;
    FCCU.NCFS[1].R = 0xFFFFFFFF;
    /* wait for the completion of the CLEAR FLAGS operation */
    while(FCCU.CTRL.B.OPS != TRANS_SUCCESSFUL) { asm("nop"); };

    FCCU.NCFK.R = 0xAB3498FE;
    FCCU.NCFS[2].R = 0xFFFFFFFF;
    /* wait for the completion of the CLEAR FLAGS operation */
    while(FCCU.CTRL.B.OPS != TRANS_SUCCESSFUL) { asm("nop"); };

    FCCU.NCFK.R = 0xAB3498FE;
    FCCU.NCFS[3].R = 0xFFFFFFFF;
    /* wait for the completion of the CLEAR FLAGS operation */
    while(FCCU.CTRL.B.OPS != TRANS_SUCCESSFUL) { asm("nop"); };

}


void FCCU_STATUS_CHK(vint32_t State_exp)
{
    /* check if FCCU is in the config state */
    /* Key for readingthe FCCU STATE: operation OP3 */
    FCCU.CTRL.R = FCCU_OP3 ;
    /* wait for the completion of the operation */
    while(FCCU.CTRL.B.OPS != 0x3) { asm("nop"); };

    if (State_exp !=FCCU.STAT.B.STATUS) {
        /* device no in config state */
        while(1) { asm("nop"); };
    }

}



void FCCU_TransToConfig(void)
{
    /* Key for the operation OP1 */
    FCCU.CTRLK.R = 0x913756AF;

    FCCU.CTRL.R = 0x1;

    /* wait for the completion of the operation */
    while(FCCU.CTRL.B.OPS != TRANS_SUCCESSFUL) { asm("nop"); };
```

```
}



void FCCU_CONFIG_RF(void)
{
   /* SW recoverable fault */
   FCCU.NCF_CFG[0].R = 0xFFFFFFFF; //  CH0 - CH15
   FCCU.NCF_CFG[1].R = 0xFFFFFFFF;
   FCCU.NCF_CFG[2].R = 0xFFFFFFFF;
   FCCU.NCF_CFG[3].R = 0xFFFFFFFF;

   /* Recoverable fault state configuration
   00: No reset reaction <------------
   01: Short functional reset (FAULT state reaction)
   10: Long functional reset  (FAULT state reaction)
   11: No reset reaction
   */
   FCCU.NCFS_CFG[0].R = 0x0;
   FCCU.NCFS_CFG[1].R = 0x0;
   FCCU.NCFS_CFG[2].R = 0x0;
   FCCU.NCFS_CFG[3].R = 0x0;
   FCCU.NCFS_CFG[4].R = 0x0;
   FCCU.NCFS_CFG[5].R = 0x0;
   FCCU.NCFS_CFG[6].R = 0x0;
   FCCU.NCFS_CFG[7].R = 0x0;

   /* Recoverable fault enable
   0: No actions following the respective recoverable fault assertion
   1: FCCU moves to ALARM or FAULT state

   Enabled RF are:
   SWT_2 ( (RF 12) */

   FCCU.NCFE[0].R= (1 << FCCU_RF_SWT_2);
   FCCU.NCFE[1].R= 0x0;
   FCCU.NCFE[2].R= 0x0;
   FCCU.NCFE[3].R= 0x0;

   /* Recoverable fault time-out enable
   0: FCCU moves into the FAULT state if the respective fault is enabled
(NCFEx is set)
   1: FCCU moves into the ALARM state if the respective fault is enabled
(NCFEx is set)
   */
```

```
        FCCU.NCF_TOE[0].R = (1 << FCCU_RF_SWT_2);

        FCCU.NCF_TO.R= 0x000003C0;

        /* Interrupt enable for all failures */
        FCCU.IRQ_ALARM_EN[0].R = (1 << FCCU_RF_SWT_2);

        /* Error out signalling enabled for all failures */
        FCCU.EOUT_SIG_EN[0].R = 0xFFFFFFFF;
        FCCU.EOUT_SIG_EN[1].R = 0xFFFFFFFF;
        FCCU.EOUT_SIG_EN[2].R = 0xFFFFFFFF;
        FCCU.EOUT_SIG_EN[3].R = 0xFFFFFFFF;

        FCCU.CFG.B.FCCU_SET_AFTER_RESET = 0x1; /* error out pin enabled */
        FCCU.CFG.B.FOM = 0x1;/* TIME SWITCHING PROTOCOL */
}




/**************************************************************************
******
 * Function Name : FCCU_TransToNormal
 * Description   : FCCU moves in NORMAL state
 * Input         : None
 * Output        : None

***************************************************************************
*****/
void FCCU_TransToNormal(void)
{
        /* Key for the operation OP2 */
        FCCU.CTRLK.R = 0x825A132B;
        /* Set the FCCU into the NORMAL state [OP2] */
        FCCU.CTRL.R = FCCU_OP2;
        //W32(FCCU.CTRL.R, 0x2);

        /* wait for the completion of the operation */
        while(FCCU.CTRL.B.OPS != TRANS_SUCCESSFUL) { asm("nop"); };


}


/**************************************************/
```

```
/* This function logs the status of the FCCU
/* in an user defined record
/*************************************************/
void FCCU_StatusLog(tuFCCU_StatusLog *FCCU_Log) {
   /* read the status register of the FCCU FSM */
   FCCU.CTRL.R = FCCU_OP3 ;
   while(FCCU.CTRL.B.OPS != 0x3) { asm("nop"); };
   FCCU_Log->STAT = FCCU.STAT.R;

   FCCU_Log->FSM = (volatile tenFccuState) (FCCU_Log->STAT & 0x7);

   /* read the status register of the FCCU recoverable faults */
   FCCU.CTRL.R = FCCU_OP10 ;
   while(FCCU.CTRL.B.OPS != 0x3) { asm("nop"); };
   FCCU_Log->RF_S0 = FCCU.NCFS[0].R;
   FCCU_Log->RF_S1 = FCCU.NCFS[1].R;
   FCCU_Log->RF_S2 = FCCU.NCFS[2].R;
   FCCU_Log->RF_S3 = FCCU.NCFS[3].R;
}
```

Appendix B Further information

B.1 Reference document

- *SPC570Sx 32-bit Power Architecture® microcontroller for automotive ASILD applications* (RM0349, DocID024507)
- *Safety Manual for SPC570S family* (AN4247, DocID024209).

B.2 Acronyms

Table 1. Acronyms

Acronym	Name
FCCU	Fault Collection and Control Unit
FSM	Finite state machine
FOSU	FCCU output supervision unit
IRQ	Interrupt request
EOUT	Error out
NCF(RF)	Non critical fault/Recoverable fault
CF(UF)	Critical fault/Unrecoverable fault
STCU	Self-Test Control Unit
MEMU	Memory Error Management Unit

Revision history

Table 1. Document revision history

Date	Revision	Changes
27-Oct-2015	1	Initial release.