AN4124
Application note

Using SPC56EL60x fault collection and control unit (FCCU)

Introduction

This application note describes in detail how to use the main features of the SPC56EL60x fault collection and control unit module (FCCU).

The fault collection and control unit offers a redundant hardware channel to collect errors and, as soon as a failure is detected, to lead the device to a safety state in a controlled way. No CPU intervention is required for collection and control operation.

The FCCU circuitry is checked at start-up (after boot) by the self-checking procedure. The FCCU is operative with a default configuration (without CPU intervention) immediately after the completion of the self-checking procedure.

Two classes of faults are identified based on the criticity and the related reactions.

Internal (that is, short or long functional reset, interrupt request) and external (EOUT signaling) reactions are statically defined or programmable based on the fault criticity.

The default configuration can be modified only in a specific FCCU state for application/test/debugging purposes.

Contents

List of tables

List of figures

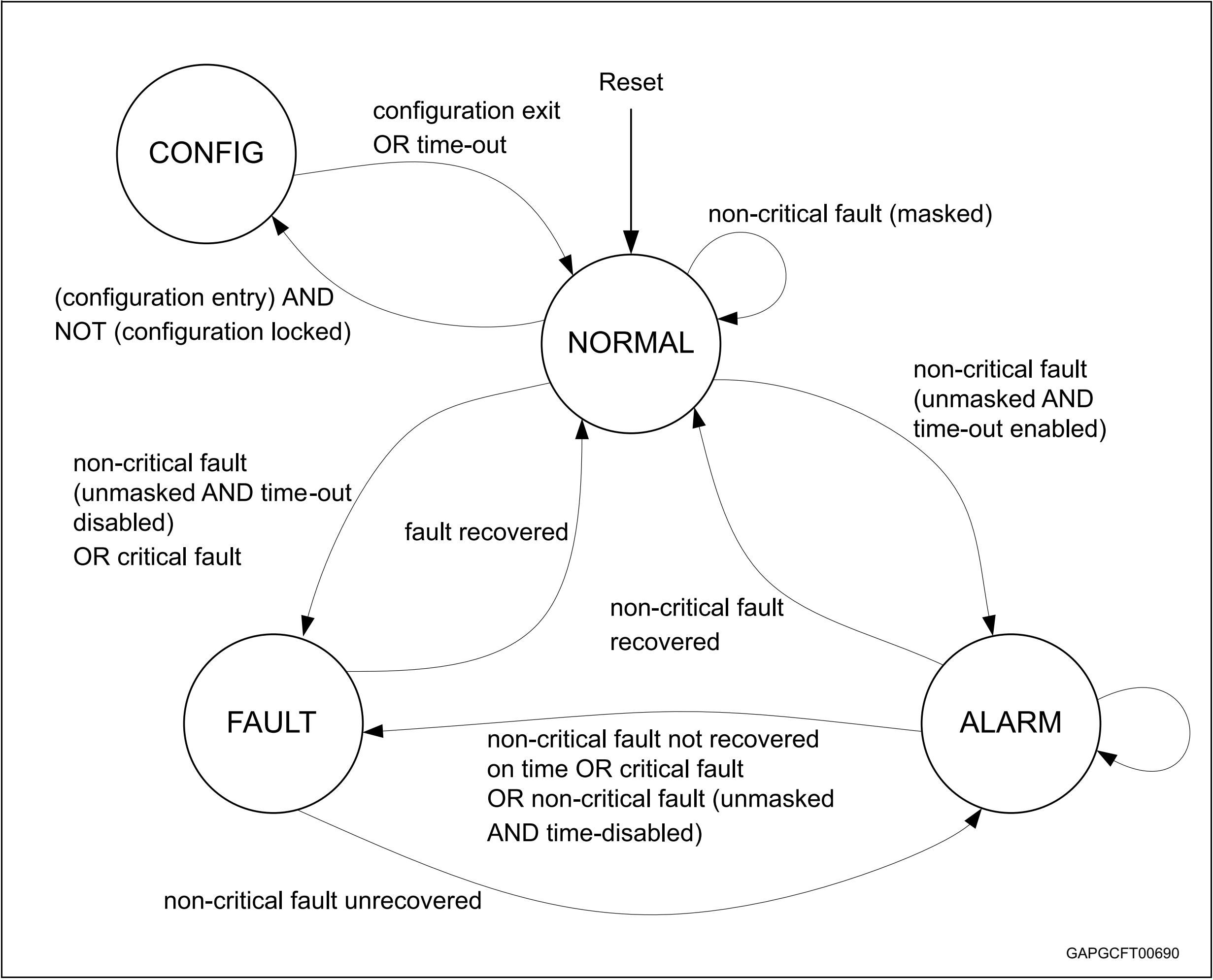
1 FCCU main features

The FCCU features are:

- The fault control and collection unit (FCCU) is a hardware IP providing a central capability to control and collect faults reported by individual modules of the SoC.
- Faults are reported to the outside world via output pin(s), if no recovery is provided by SoC. No internal actions (such as IRQ, Reset) can be taken.
- The operation of the fault collection unit is independent of the CPU, so the FCCU provides a fault reporting mechanism even if the CPU is malfunctioning.
- The fault control and collection unit is developed specifically to increase the level of the safety of the system and ECU.
- The FCCU allows a redundant path to the RGM to enter failsafe mode in case of error.

Below *Figure 1* FCCU-SM (state machine):

Figure 1. FCCU state machine



2 HW/SW recoverability fault

In general, the following definitions are applicable to fault management:

- HW recoverable fault: the fault indication is a level sensitive signal that is asserted until the cause of the fault is removed. Typically the fault signal is latched in an external module to the FCCU. The FCCU state transitions are consequently executed on the state changes of the input fault signal (fccu_cf[] or fccu_ncf[]). No SW intervention in the FCCU is required to recover the fault condition.
- SW recoverable fault: the fault indication is a signal asserted without a defined time duration. The fault signal is resynchronized and latched in the FCCU. The fault recovery is executed following a SW recovery procedure (status/flag register clearing).

The following types of reset are applicable:

- Destructive reset: any type of reset related to a power failure condition that implies a complete system reinitialization
- Long functional reset: implies FLASH and digital circuitry (with some exceptions, including FCCU, STCU) initialization
- Short functional reset: implies digital circuitry (with some exceptions, including FCCU, STCU) initialization

3 Fault dual path: FCCU and RGM

Due to the dual path, many faults (critical and not) reach the RGM and FCCU.

NMI can be mapped in RAM. For this reason the NMI is cleared after RESET condition.

In general, when a fault occurs, if it is mapped on RGM and FCCU, the RGM generates a RESET, independently of FCCU settings. After RESET (generated by fault) the system is in SAFE state. Looking in the FCCU CFSx status register (by procedure), it may recognize the fault, and react to it. After fault recovery the system transition can be: SAFE => RUN.

If the fault is mapped only on FCCU (as CF[20]), when it occurs the system resets or generates an NMI assertion, depending on FCCU settings. In RESET case, the FCCU generates a RESET by RGM.

Since the system stays in SAFE state, it does nothing. After the system transitions from SAFE to RUN (and fault is set), the system unmasks the NMI. By NMI ISR it is possible to clear the fault state registers.

3.1 RGM module

The reset generation module (MC_RGM) centralizes the different reset sources and manages the reset sequence of the device.

Table 1 shows the RGM_FES (Functional Event Status) register bitmap.

Table 1. RGM_FES register

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_CLR	F_PLL0	F_CWD	F_SOFT	F_CORE	F_JTAG
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4 Fault: CF and NCF

The FAULT state has a higher priority than the ALARM state, in the case of concurrent fault events (critical and non-critical) that occur in the NORMAL state. In case of concurrent critical faults, the fault reaction corresponds to the worst case (for example, a long functional reset is asserted if it has been programmed).

The ALARM to FAULT state transition occurs if a fault (unmasked and with time-out disabled) is asserted in the ALARM state.

Any critical fault (programmed to react with a hard or soft reaction) that occurs when the FCCU is already in the FAULT state causes an immediate hard or soft reaction (long or short functional reset).

The ALARM to NORMAL state transition occurs only if all the non-critical faults (including the faults that have been collected after entry to ALARM state) have been cleared (SW or HW recovery). Otherwise the FCCU will remain in the ALARM state.

The FAULT to NORMAL state transition occurs only if all the critical and non-critical faults (including the faults that have been collected after entry to FAULT/ALARM state) have been cleared (SW or HW recovery). Otherwise the FCCU remains in the FAULT state (if any critical fault is still pending) or returns to the ALARM state (if any non-critical fault is still pending and the time-out has not elapsed).

4.1 Critical fault (CF)

Below is the CF table:

Table 2. Critical fault

Critical fault	Source	Signal	Short/long/none default func. reset	Set/clear injection
CF[0]	RCCUO[0]	rcc_out	Long	X
CF[1]	RCCU1[0]	rcc_out	Long	X
CF[2]	RCCUO[1]	rcc_out	Long	X
CF[3]	RCCU1[1]	rcc_out	Long	X
CF[4]	RCCUO[2]	rcc_out	Long	X
CF[5]	RCCU1[2]	rcc_out	Long	X
CF[6]	RCCUO[3]	rcc_out	Long	X
CF[7]	RCCU1[3]	rcc_out	Long	X
CF[8]	RCCUO[4]	rcc_out	Long	X
CF[9]	RCCU1[4]	rcc_out	Long	X
CF[10]	RCCUO[5]	rcc_out	Long	X
CF[11]	RCCU1[5]	rcc_out	Long	X
CF[12]	RCCUO[6]	rcc_out	Long	X

Table 2. Critical fault (continued)

Critical fault	Source	Signal	Short/long/none default func. reset	Set/clear injection
CF[13]	RCCU1[6]	rcc_out	Long	X
CF[14]	SWT_0	Software watchdog timer	Long	—
CF[15]	SWT_1	Software watchdog timer	Long	—
CF[16]	MCM_NCE_0	ECC not correctable error	Long	—
CF[17]	MCM_NCE_1	ECC not correctable error	Long	—
CF[18]	ADC_CF_0	Internal self test (critical fault)	—	X (by ADC itself)
CF[19]	ADC_CF_1	Internal self test (critical fault)	—	X (by ADC itself)
CF[20]	STCU_CF	Bist results (critical faults)	—	X
CF[21]	LVD_HVD_ 1.2V	LVD/HVD BIST failure result in test mode	—	X
CF[22]	SSCM_XFER_ERR	SSCM transfer error (during the STCU config loading)	—	—
CF[23]	LSM_DPM_ERR0	LSM <-> DPM runtime switch	Long	X
CF[24]	LSM_DPM_ERR1	LSM <-> DPM runtime switch	Long	X
CF[25]	—	—	—	—
CF[26]	—	—	—	—
CF[27]	STCU	STCU fault condition (run in application mode)	Long	—
CF[28]	DFT0	Combination of safety critical signals from Test Control Unit (TCU)	Long	—
CF[29]	DFT1	Combination of safety critical signals from Test Control Unit (TCU)	Long	—
CF[30]	DFT2	Combination of safety critical signals from Test Control Unit (TCU)	Long	—
CF[31]	DFT3	Combination of safety critical signals from Test Control Unit (TCU)	Long	—

4.2 Non-critical fault (NCF)

Table 3 is about the NCF table:

Table 3. Non-critical fault

Non-critical fault	Source	Signal	Short/long/ none default func reset	Fault management	Polarity	Set/clear injection
NCF[0]	Core_0 watchdog	p_wrs[0]	Long	latched	High	—
NCF[1]	Core_0 watchdog	p_wrs[1]	Long	latched	High	—
NCF[2]	FM_PLL_0	Loss of lock	Long	latched	High	—
NCF[3]	FM_PLL_1	Loss of lock	Long	latched	High	—
NCF[4]	CMU_0	Loss of XOSC clock	Long	latched	High	—
NCF[5]	CMU_0	Sysclk frequency out of range	Long	latched	High	—
NCF[6]	CMU_1	MOTC_CLK frequency out of range	Long	latched	High	—
NCF[7]	CMU_2	FRPE_CLK frequency out of range	Long	latched	High	—
NCF[8]	MCM_ECN_0	ECC 1-bit error correction notification	—	latched	High	—
NCF[9]	MCM_ECN_1	ECC 1-bit error correction notification	—	latched	High	—
NCF[10]	ADC_NCF_0	Internal self test (Non-critical fault)	—	latched	High	X (by ADC itself)
NCF[11]	ADC_NCF_1	Internal self test (Non-critical fault)	—	latched	High	X (by ADC itself)
NCF[12]	STCU_NCF	Bist results (Non-critical faults)	—	latched	High	X
NCF[13]	LVD_ 1.2V	LVD BIST OK in test mode/ LVD NOK in user mode	—	latched	High	X
NCF[14]	HVD_ 1.2V	HVD BIST OK in test mode/ HVD NOK in user mode	—	latched	High	X
NCF[15]	LVD VREG	LVD VREG fault detected by self-checking	—	latched	High	X
NCF[16]	LVD FLASH	LVD FLASH fault detected by self-checking	—	latched	High	X
NCF[17]	LVD IO	LVD IO fault detected by self-checking	—	latched	High	X
NCF[18]	PMU	Comparator fault detected by self-checking	—	latched	High	—
NCF[19]	FLEXR_ECN	ECC 1-bit error correction notification	—	latched	High	—

Table 3. Non-critical fault (continued)

Non-critical fault	Source	Signal	Short/long/ none default func reset	Fault management	Polarity	Set/clear injection
NCF[20]	FLEXR_NCE	ECC not correctable error	—	latched	High	—
NCF[21]	MC_ME	Software device reset	—	latched	High	—
NCF[22]	BP_BALLAST0	Bypass Ballast0	—	latched	High	—
NCF[23]	BP_BALLAST1	Bypass Ballast1	—	latched	High	—
NCF[24]	BP_BALLAST2	Bypass Ballast2	—	latched	High	—
NCF[25]	—	—	—	—	—	—
NCF[26]	—	—	—	—	—	—
NCF[27]	—	—	—	—	—	—
NCF[28]	—	—	—	—	—	—
NCF[29]	—	—	—	—	—	—
NCF[30]	—	—	—	—	—	—
NCF[31]	—	—	—	—	—	—

5 FCCU settings

Normally the FCCU is configured at start up. In any case, it is possible to manage some registers only in CONFIG state (according to IP Specification Block guide).

5.1 Example 1: FCCU critical fault injection (no NMI assertion)

We show the FCCU functionality by means of an example which uses fault injection (with fake funtionality), in order to show the FCCU reaction. The example is without NMI assertion. The fault is checked and cleared by looking in the CFSx registers.

Example description

- Put FCCU in CONFIG state: set registers
- Return to NORMAL state by means of a procedure or by allowing timer out to elapse
- Inject (fake) faults
- After RESET (by RGM), verify, in SAFE state (without NMI), which fault was detected (FCCU_CFS0 register)
- Clear the FCCU_CFS0 register (by suitable procedure)

Example procedure

- After reset the FCCU automatically enters NORMAL state.
- Configure FCCU in CONFIG with Dual-rail Encoding Protocol.
  - Write the key to the FCCU_CTRLK register [OP1].
  - Write the FCCU_CTRL register (operation OP1).
- Emulate all (fake) SW/HW faults.
  - FCCU_CFG: (Configuration Register)
  - SM = 1 (EOUT protocol (dual-rail, time-switching) fast switching mode)
  - PS = 1 (fcc_eout[1] active low, fcc_eout[0] active low)
  - FOM = 000 (Fault Output Mode selection = Dual-Rail (default state) [fccu_eout[1:0] = outputs])
  - FOP=0 (Fault Output Prescaler = Input clock frequency (ipg_clk_safe clock) is divided by 2048)
- Enter NORMAL state.
  - Write the key into the FCCU_CTRLK register [OP2].
  - Write the FCCU_CTRL register (operation OP2).
- Set fault by FCCU_CFF registers (RESET assertion by RGM).
- Read and verify FCCU_CFS0..3 by means of procedure (NMI was masked).
- Clear HW/SW faults from FCCU_CFS0..3 by means of procedure.

Code

After the core initialization in main function the code is (NMI masked):

```
if (ME.GS.B.S_CURRENT_MODE == 2){    /* SAFE MODE */
    if((FCCU_Clear_CRITICAL_Fault()) == PASS){
          /* Test PASS */
    }else{
          while(1);         /* Test FAIL */
    }
}else{                /* DRUN MODE */
    /* ----------------- Test INIT --------------------- */
    if((tc0_INIT()) == PASS){
          /* ----------------- Fake Fault --------------------- */
          FCCU.CFF.R = 0;   /* First Fault injection */
    }else{
        while(1);     /* Test FAIL */
    }
    /* -----------------END  Test INIT --------------------- */
}
```

Description

At the beginning the microcontroller is in DRUN mode, the “else” condition is asserted, by tc0_INIT procedure. In the tc0_INIT, the FCCU will be configured. When the injection fault is asserted (FCCU.CFF.R = 0), the system will reset. After the start up, in main function the system is in SAFE mode (the NMI is masked). Then the “if” condition is asserted, and all faults are cleared.

5.1.1 FCCU init

```
uint16_t tc0_INIT(void){
    /* ----------------- CONFIG State --------------------- */
    FCCU_CONFIG_STATE();/* CONFIG state */
    FCCU.CFG.B.SM = 1; /* EOUT protocol (dual-rail, time-switching)
                            fast switching mode*/
    FCCU.CFG.B.PS = 1; /* fcc_eout[1] active low, fcc_eout[0] active low */
    FCCU_CFG_FOM_Config(CFG_FOM0); /* CFG_FOM0 = Dual-Rail (default state)
                            [fccu_eout[1:0]= outputs] */
    FCCU_CFG_FOP_Config(0); * Fault Output Prescaler= Input clock frequency
                            (ipg_clk_safe clock) is divided by 2 x 1024 */

    /* ----------------- NORMAL State --------------------- */
     FCCU_NORMAL_STATE();    /* NORMAL state */
    return(PASS);
}
```

5.2 Example 2: FCCU critical fault injection (NMI assertion)

In this example we show the fault injection (with fake funtionality), in order to show an FCCU reaction. The example is with NMI assertion. The fault is checked and cleared inside the NMI subroutine by looking in the CFSx registers.

Example description

- Put FCCU in CONFIG state: set registers.
- Return to NORMAL state: by means of procedure or by allowing timer out to elapse.
- Inject (fake) faults.
- Verify that in SAFE state (NMI management), and that fault is detected (FCCU_CFS0 register).
- Clear the FCCU_CFS0 register (by suitable procedure).

Example procedure

- After the reset the FCCU automatically enters NORMAL state.
- Configure FCCU in CONFIG with Dual-rail Encoding Protocol.
  - Write the key to the FCCU_CTRLK register [OP1].
  - Write the FCCU_CTRL register (operation OP1).
- Emulate all (fake) SW/HW faults.
  - FCCU_CFG_TO = 0x7 (Set Timer Out)
  - FCCU_CFG: (Configuration Register)
  - SM = 1 (EOUT protocol (dual-rail, time-switching) fast switching mode)
  - PS = 1 (fcc_eout[1] active low, fcc_eout[0] active low)
  - FOM = 000 (Fault Output Mode selection = Dual-Rail (default state) [fccu_eout[1:0] = outputs])
  - FOP = 0 (Fault Output Prescaler= Input clock frequency (ipg_clk_safe clock) is divided by 2048)
  - FCCU_CFS_CFG0 = 0 (No reset reaction)
- Enter NORMAL state.
  - Write the key to the FCCU_CTRLK register [OP2].
  - Write the FCCU_CTRL register (operation OP2).
- Set fault by FCCU_CFF registers (no RESET assertion).
- NMI assertion: NMI_ISR managing
- Read and verify FCCU_CFS0..3 by means of procedure.
- Clear HW/SW faults from FCCU_CFS0..3 by means of procedure.

Code

After the core initialization in main function the code is:

```
if((tc0_INIT()) == PASS){
   /* ----------------- Fake Fault --------------------- */
   FCCU.CFF.R = 20;      /* N. 20 Fault injection */
   Delay(10000);         /* Dealay */
}else{
   /* tc0_INIT - FAILURE */
}
```

Description

First the microcontroller is in DRUN mode, the “if” condition is asserted, by the tc0_INIT procedure. The FCCU is configured in the tc0_INIT. Next the CF 20 is injected (in order to generate an NMI, without RESET). When the injection fault is asserted (FCCU.CFF.R = 20), the system asserts NMI. In NMI ISR the fault is cleared and the system enters RUN mode.

5.2.1 FCCU init

```
uint16_t tc0_INIT(void){
/* ----------------- CONFIG State --------------------- */
    FCCU.CFG_TO.R = 0x7;/* Set Timer Out CCONFIG STATE to 8.192 ms */
    FCCU_CONFIG_STATE();/* CONFIG state */
    FCCU.CFG.B.SM = 1; /* EOUT protocol (dual-rail, time-switching) fast switching
                  mode*/
    FCCU.CFG.B.PS = 1; /* fcc_eout[1] active low, fcc_eout[0] active low */
    FCCU_CFG_FOM_Config(CFG_FOM0); /* CFG_FOM0 = Dual-Rail (default state)
                  [fccu_eout[1:0]= outputs] */
    FCCU_CFG_FOP_Config(0); /* Fault Output Prescaler= Input clock frequency
                  (ipg_clk_safe clock) is divided by 2 x 1024 */

    /* Set the Critical Fault reaction */
    FCCU.CFS_CFG0.R = 0; /* No reset reaction */

    /* ----------------- NORMAL State --------------------- */
    FCCU_NORMAL_STATE();/* NORMAL state */
    return(PASS);
}
```

5.3 Example 3: FCCU - Non-critical fault injection

In this example we show fault injection (with fake funtionality), in order to show an FCCU reaction. The example is with FAULT_ISR assertion. The fault is checked and cleared in FAULT_ISR subroutine by looking in the CFSx registers.

Example description

- Put FCCU in CONFIG state: set registers.
- Return to NORMAL state: by means of procedure or by allowing timer out to elapse.
- Inject (fake) faults (NCF N. 12).
- Verify that in RUN state (FAULT_ISR management), and that fault is detected (FCCU_NCFS0 register).
- Clear the FCCU_NCFS0 register (by suitable procedure).

Example procedure

- After the reset the FCCU automatically enters NORMAL state.
- Configure FCCU in CONFIG with Dual-rail Encoding Protocol.
  - Write the key to the FCCU_CTRLK register [OP1].
  - Write the FCCU_CTRL register (operation OP1).
- Emulate all (fake) SW/HW faults.
  - FCCU_CFG_TO=0x7 (Set Timer Out)
  - FCCU_CFG: (Configuration Register)
  - SM=1 (EOUT protocol (dual-rail, time-switching) fast switching mode)
  - PS=1 (fcc_eout[1] active low, fcc_eout[0] active low)
  - FOM=000 (Fault Output Mode selection= Dual-Rail (default state) [fccu_eout[1:0]= outputs])
  - FOP=0 (Fault Output Prescaler= Input clock frequency (ipg_clk_safe clock) is divided by 2 x 1024)
  - FCCU_NCFS_CFG0 = 0 (No reset reaction)
  - FCCU_NCFE0 = 0xFFFFFFFF; (Enable FCCU to move to ALARM or FAULT State)
  - FCCU_NCF_TOE0 = 0xFFFFFFFF; (FCCU moves into the ALARM state if the respective fault is enabled (NCFEx is set))
  - FCCU_NCF_TO = 0xFFFFFFFF; (Non-critical fault time-out)
- Enter NORMAL state.
  - Write the key to the FCCU_CTRLK register [OP2].
  - Write the FCCU_CTRL register (operation OP2).
- Set fault by FCCU_NCFF registers (NCF N. 12).
- ISR assertion: FAULT_ISR managing (ISR N. 250)
- Read and verify FCCU_NCFS0..3 by means of procedure.
- Clear HW/SW faults from FCCU_NCFS0..3 by means of procedure.

Code

After the core initialization in main function the code is:

```
if((tc1_INIT()) == PASS){
   /* ----------------- Fake NCF Fault --------------------- */
   FCCU.NCFF.R = 12;    /* N. 12 NCF Fault injection */
   Delay(10000);        /* Delay */
}else{
   /* tc1_INIT - FAILURE */
}
```

Description

At the beginning the micro is in DRUN mode. The FCCU is configured in the tc1_INIT. Next the NCF 12 is set. When the injection fault is asserted (FCCU.NCFF.R = 12), the system asserts FAULT ISR. In FAULT ISR the fault is cleared and the system enters RUN mode.

5.4 LOCK FCCU configuration

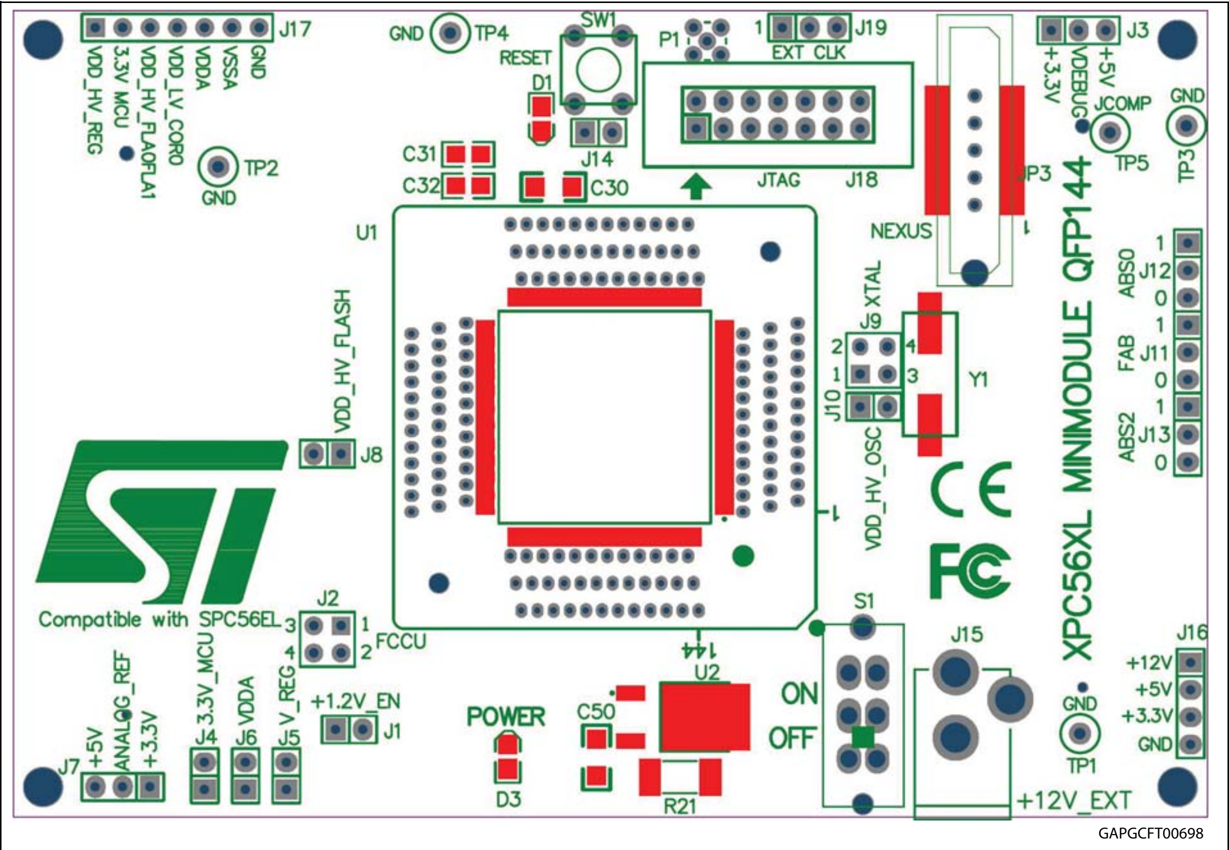
The configuration state is used to modify the default configuration of the FCCU only. A sub-set of the FCCU registers, dedicated to defining the FCCU configuration (global configuration, reactions to fault, time-out, non-critical fault masking), can be accessed in write mode, in the CONFIG state only.

The CONFIG state is accessible in NORMAL state only and only if the configuration is not locked. The configuration lock can be disabled only by a global reset of the FCCU. To lock the FCCU see *Section A.2.3* 0 function.

5.5 Hardware: XPC56XL minimodule

The examples are executed by XPC56EL minimodule, using the motherboard. The motherboard provides common functionality used in most applications, such as serial communication interface, CAN transceivers, SPI bus, I/O pins, power supply, buttons and LEDs. The minimodule provides a minimum setup for the microprocessor, for example, socket for the processor, crystal oscillator and debug interface. *Figure 2* displays the XPC56XL minimodule layout.

Figure 2. XPC56EL minimodule



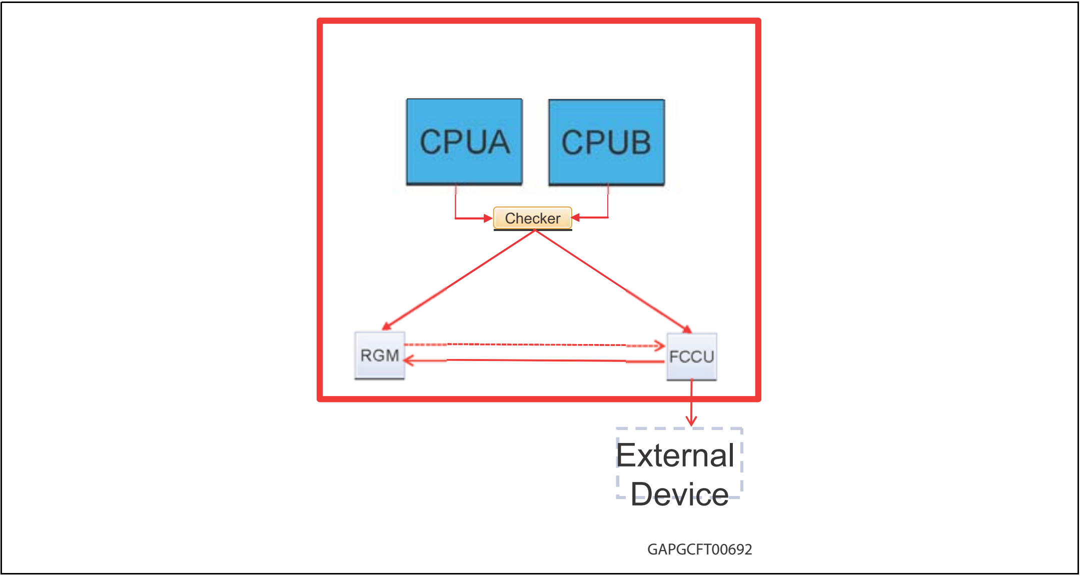
Appendix A Redundancy and functions

A.1 Path redundancy on critical error reaction

All faults detected are reported to the central Fault Collection and Control Unit (FCCU) and RGM. Depending on the particular fault, the FCCU puts the device into the appropriate configured Fail-Safe state. This prevents propagation of faults to system level.

Critical errors detected normally are forwarded independently by each channel to RGM and FCCU. The state of the RGM is forwarded to the FCCU. The FCCU forwards an additional reset request to the RGM. This strategy is used as it drastically decreases the common mode failure on the Reset path.

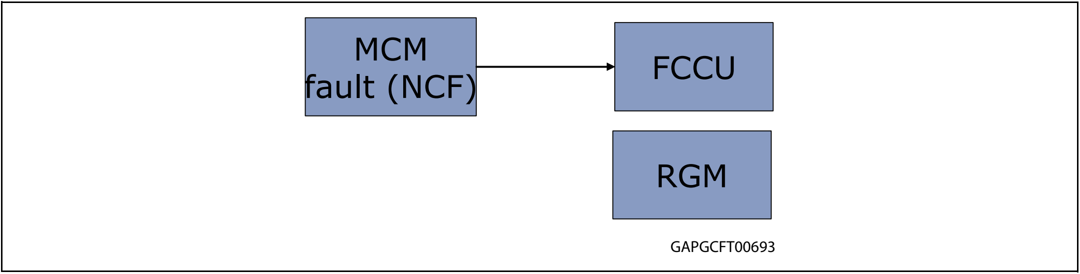
Figure 3. Dual path faults



For some faults:

- The fault is triggered to the FCCU.
- FCCU reacts independently of the RGM.
- Fault reaction depends on the FCCU settings.

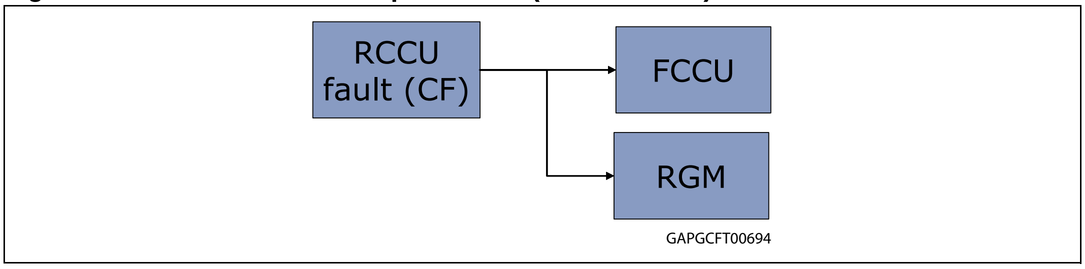
Figure 4. RGM/FCCU – no dual path faults



For some faults (Critical Faults):

- RGM and FCCU react to the fault independently.
- RGM resets the device (LR).
  - FCCU is not reset by RGM reset.
- FCCU takes some action depending on the configuration.
  - FCCU signals the fault externally.
  - after the reset the device enters SAFE mode.
  - NMI

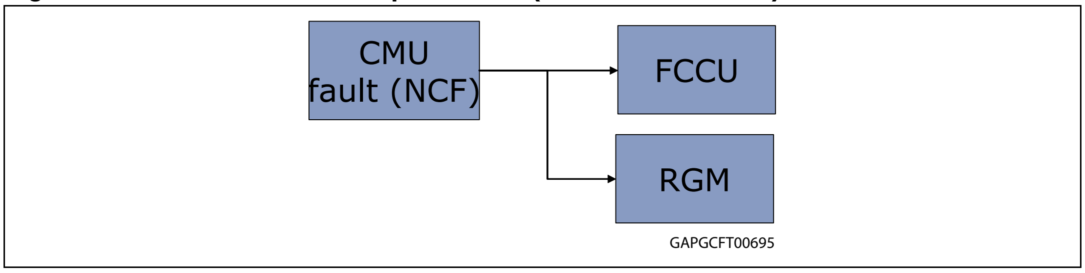
Figure 5. RGM/FCCU – dual path faults (critical faults)



For some faults (Non-critical Faults):

- RGM and FCCU react to the fault independently/
- RGM reaction is configurable.
  - IRQ
- FCCU takes some action depending on the configuration.
  - FCCU waits for the NCF timeout.
  - FCCU signals the fault externally.
  - device enters SAFE mode
  - NMI

Figure 6. RGM/FCCU – dual path faults (Non-critical Faults)



A.2 General purpose function

Below are some general purpose functions, settinf and clearing registers.

A.2.1 Config state

```
uint32_t FCCU_CONFIG_STATE(void){

    /* ----------------- CONFIG State --------------------- */
    FCCU.CTRLK.R = CTRLK_OP1;       /* Key for the operation OP1 */
    FCCU.CTRL.R = CTRL_OPR1;        /* Set the FCCU into the CONFIG state [OP1] */
    while(FCCU.CTRL.B.OPS != CTRL_OPS3);     /* wait for the completion of the
                           operation */

    return 1;
}
```

A.2.2 Normal state

```
uint32_t FCCU_NORMAL_STATE(void){

    /* ----------------- NORMAL State --------------------- */
    FCCU.CTRLK.R = CTRLK_OP2;       /* Key for the operation OP2 */
    FCCU.CTRL.R = CTRL_OPR2;        /* Set the FCCU into the NORMAL state [OP2] */
    while(FCCU.CTRL.B.OPS != CTRL_OPS3);     /* wait for the completion of the
                           operation */
    return 1;
}
```

A.2.3 Lock FCCU

```
uint32_t FCCU_LOCK(void){

    /* ----------------- NORMAL State --------------------- */
    FCCU.CTRLK.R = CTRLK_OP16;      /* Key for the operation OP16 */
    FCCU.CTRL.R = CTRL_OPR2;        /* Lock the FCCU configuration [OP16] */
    while(FCCU.CTRL.B.OPS != CTRL_OPS3);     /* wait for the completion of the
                           operation */
    return 1;
}
```

A.2.4 Read status register

```
uint32_t FCCU_CFS_Read(uint32_t CFS_number, uint32_t* CFS_Value){
    uint32_t exit_value= 0;/* Returned value = ERROR */
    uint32_t Reg_Selection = 0;/* Register Selection [0..3] */

    if (CFS_number <= 127){
         FCCU.CTRL.B.OPR = CTRL_OPR9;   /* Set the OP9 */
         while(FCCU.CTRL.B.OPS != CTRL_OPS3); /* wait for the completion of the
                  operation */
         Reg_Selection = (CFS_number/32);   /* INT(CFS_number/32)*/
         switch (Reg_Selection){
                  case 0 : *CFS_Value = (FCCU.CFS0.R >> (CFS_number%32)) & 1;
                           /* Read the critical fault latched state */
                           break;
                  case 1 : *CFS_Value = (FCCU.CFS1.R >> (CFS_number%32)) & 1;
                           /* Read the critical fault latched state */
                           break;
                  case 2 : *CFS_Value = (FCCU.CFS2.R >> (CFS_number%32)) & 1;
```

```
                                /* Read the critical fault latched state */
                                break;
                       case 3 : *CFS_Value = (FCCU.CFS3.R >> (CFS_number%32)) & 1;
                                /* Read the critical fault latched state */
                                break;
                       default: *CFS_Value = (FCCU.CFS3.R >> (CFS_number%32)) & 1;
                                /* Read the critical fault latched state */
                                break;
              }
              exit_value = 1;            /* Returned value = SUCCESS */
         }
         else {
             /* ERROR*/
         };
         return(exit_value);
     }
```

A.2.5 Clear fault

```
uit32_t FCCU_CFS_Clear(uint32_t CFS_number){
    uint32_t exit_value= 0;          /* Returned value = ERROR */
    uint32_t Reg_Selection = 0;      /* Register Selection [0..3] */
    uint32_t Support = 0;            /* Support variable */
    uint32_t CFS_Value;

    if (CFS_number <= 127){
    Reg_Selection = (CFS_number/32);   /* INT(CFS_number/32)*/
    switch (Reg_Selection){
        case 0 : Support = FCCU.CF_CFG0.R;
               break;
        case 1 : Support = FCCU.CF_CFG1.R;
               break;
        case 2 : Support = FCCU.CF_CFG2.R;
               break;
        case 3 : Support = FCCU.CF_CFG3.R;
               break;
        default: Support = FCCU.CF_CFG3.R;
               break;
    }
    Support = (Support >> (CFS_number%32)) & 0x1;
    if (Support ==  CFG_SW){  /* SW recoverable fault*/
        do{
          switch (Reg_Selection){
              case 0 : FCCU.CFK.R = CFK_Key;  /* set the Critical fault key */
                         FCCU.CFS0.R = (uint32_t) (1 << (CFS_number%32));
                             /* reset the critical fault state */
                         break;
              case 1 : FCCU.CFK.R = CFK_Key;  /* set the Critical fault key */
                         FCCU.CFS1.R = (uint32_t) (1 << (CFS_number%32));
                             /* reset the critical fault state */
                         break;
              case 2 : FCCU.CFK.R = CFK_Key;  /* set the Critical fault key */
                         FCCU.CFS2.R = (uint32_t) (1 << (CFS_number%32));
                             /* reset the critical fault state */
                         break;
              case 3 : FCCU.CFK.R = CFK_Key;  /* set the Critical fault key */
                         FCCU.CFS3.R = (uint32_t) (1 << (CFS_number%32));
                             /* reset the critical fault state */
                         break;
              default: FCCU.CFK.R = CFK_Key;  /* set the Critical fault key */
                         FCCU.CFS3.R = (uint32_t) (1 << (CFS_number%32));
                             /* reset the critical fault state */
                         break;
          }
          while(FCCU.CTRL.B.OPS != CTRL_OPS3); /* wait for the completion of
                   the operation */
          if ( FCCU_CFS_Read(CFS_number, &CFS_Value)){
              if (CFS_Value == 0){
                   exit_value = 1;   /* Returned value = SUCCESS */
              }
          }
        }while(CFS_Value==1);
    }
    else {
        /* HW recoverable fault*/
    }
    };
    return(exit_value);
}
```

A.2.6 Clear all critical faults

```
uint16_t FCCU_Clear_CRITICAL_Fault(void){
    tU32 CFS_Value;
    uint8_t tc0_error = 0;/* Error counter */

    for(Num_Fault = 0;Num_Fault <= 24; Num_Fault++){   /* Num_Fault <= 24 */
    /* ----------------- Read State --------------------- */
    if(FCCU_CFS_Read(Num_Fault, &CFS_Value)){
        if (CFS_Value == 1){/* The fault was latched correctly */
                if((RGM.FES.R & 0x0080) == 0x0080){
                      /* Retun from FCCU SAFE mode reset */
                      FCCU_CFS_Clear(Num_Fault);   /* Clear the fault by procedure */
                      RGM.FES.R = 0xFFFF; /* Clear FER register */
                      ME.MCTL.R = (DRUN_MODE << 28 | 0x00005AF0); /* Mode & Key */
                      ME.MCTL.R = (DRUN_MODE << 28 | 0x0000A50F); /* Mode & Key *
                      /* Wait for mode entry to complete */
                      while(ME.GS.B.S_MTRANS==1);
                      /* Check DRUN mode has been entered */
                      while(ME.GS.B.S_CURRENT_MODE!=DRUN_MODE);
                      tc0_error = 0;    /* Error counter */
                }
          }else{
                /* No fault was latched */
          }
    }else{
        /* Read State ERROR */
    }
    }
    if(tc0_error == 0)
          return(PASS);
    return(FAIL);
}
```

A.2.7 Clear all Non-critical faults

```
uint16_t FCCU_Clear_NON_CRITICAL_Fault(void){
    tU32 NCFS_Value;
    uint8_t tc1_error = 0;/* Error counter */

    for(Num_Fault = 0; Num_Fault <= 24; Num_Fault++){
        /* ----------------- Read State --------------------- */
          if(FCCU_NCFS_Read(Num_Fault, &NCFS_Value)){
                if (NCFS_Value == 1){/* The fault was latched correctly */
```

```
                                    /* Retun from FCCU SAFE mode reset */
                                    FCCU_NCFS_Clear(Num_Fault);   /* Clear the fault */
                                    RGM.FES.R = 0xFFFF; /* Clear FER register */
                                    ME.MCTL.R = (DRUN_MODE << 28 | 0x00005AF0);/* Mode & Key */
                                    ME.MCTL.R = (DRUN_MODE << 28 | 0x0000A50F);/* Mode & Key */
                                    /* Wait for mode entry to complete */
                                    while(ME.GS.B.S_MTRANS==1);
                                    /* Check DRUN mode has been entered */
                                    while(ME.GS.B.S_CURRENT_MODE!=DRUN_MODE);
                                    tc1_error = 0;          /* Error counter */
                              }else{
                                    /* Not NON-Critical fault was latched */
                              }
                        }else{
                              /* Read State ERROR */
                        }
                  }
                  if(tc1_error == 0)
                  return(PASS);
                  return(FAIL);
            }
```

A.2.8 Read FCCU - state machine

```
uint32_t FCCU_STATUS_Read(uint32_t* STATUS_Value){
    uint32_t exit_value= 0; /* Returned value = ERROR */

    FCCU.CTRL.B.OPR = CTRL_OPR3;    /* Set the OP3 */
    while(FCCU.CTRL.B.OPS != CTRL_OPS3); /* wait for the completion of
                  the operation */
    *STATUS_Value = FCCU.STAT.R; /* Read the STATUS Register */
    exit_value = 1;         /* Returned value = SUCCESS */
    return(exit_value);
}
```

A.2.9 Non-critical fault - enable

```
uint32_t FCCU_NCF_Enable(uint32_t NCFE_number, uint32_t NCFE_Value){
    uint32_t exit_value= 0; /* Returned value = ERROR */
    uint32_t Reg_Selection = 0;/* Register Selection [0..3] */

    if (NCFE_number <= 127){
    Reg_Selection = (NCFE_number/32);   /* INT(NCFE_number/32)*/
    if (NCFE_Value == NCFE_En){
          switch (Reg_Selection){
                case 0 :    FCCU.NCFE0.R |= (uint32_t) (NCFE_En << (NCFE_number%32));
                            /* Enable the non-critical fault */
                            break;
                case 1 :    FCCU.NCFE1.R |= (uint32_t) (NCFE_En << (NCFE_number%32));
                            /* Enable the non-critical fault */
                            break;
                case 2 :    FCCU.NCFE2.R |= (uint32_t) (NCFE_En << (NCFE_number%32));
                            /* Enable the non-critical fault */
                            break;
                case 3 : FCCU.NCFE3.R |= (uint32_t) (NCFE_En << (NCFE_number%32));
                            /* Enable the non-critical fault */
                            break;
                default:    FCCU.NCFE3.R |= (uint32_t) (NCFE_En << (NCFE_number%32));
                            /* Enable the non-critical fault */
                            break;
        }
    }else{
          switch (Reg_Selection){
                case 0 : FCCU.NCFE0.R &= (uint32_t) ~(NCFE_En << (NCFE_number%32));
                            /* Disable the non-critical fault */
                            break;
                case 1 : FCCU.NCFE1.R &= (uint32_t) ~(NCFE_En << (NCFE_number%32));
                            /* Disable the non-critical fault */
                            break;
                case 2 : FCCU.NCFE2.R &= (uint32_t) ~(NCFE_En << (NCFE_number%32));
                            /* Disable the non-critical fault */
                            break;
          case 3 : FCCU.NCFE3.R &= (uint32_t) ~(NCFE_En << (NCFE_number%32));
                            /* Disable the non-critical fault */
                            break;
          default: FCCU.NCFE3.R &= (uint32_t) ~(NCFE_En << (NCFE_number%32));
                            /* Disable the non-critical fault */
                            break;
        }
    }
    };
    return(exit_value);
}
```

A.2.10 NCF - normal to alarm - read state

```
uint32_t FCCU_NAFS_Read(uint32_t* NAFS_Value){
    uint32_t exit_value= 0; /* Returned value = ERROR */

    FCCU.CTRL.B.OPR = CTRL_OPR4;   /* Set the OP4 */
    while(FCCU.CTRL.B.OPS != CTRL_OPS3); /* wait for the completion of the
                              operation */
    *NAFS_Value = FCCU.NAFS.R; /* Read the NAFS latched state */
    exit_value = 1;         /* Returned value = SUCCESS */
    return(exit_value);
}
```

A.2.11 NCF - normal to alarm - clear state

```
uint32_t FCCU_NAFS_Clear(void){
    uint32_t exit_value= 0; /* Returned value = ERROR */

    FCCU.CTRL.B.OPR = CTRL_OPR13;   /* Set the OP13 */
    while(FCCU.CTRL.B.OPS != CTRL_OPS3); /* wait for the completion of the
                              operation */
    exit_value = 1;         /* Returned value = SUCCESS */
    return(exit_value);
}
```

A.2.12 IRQ status

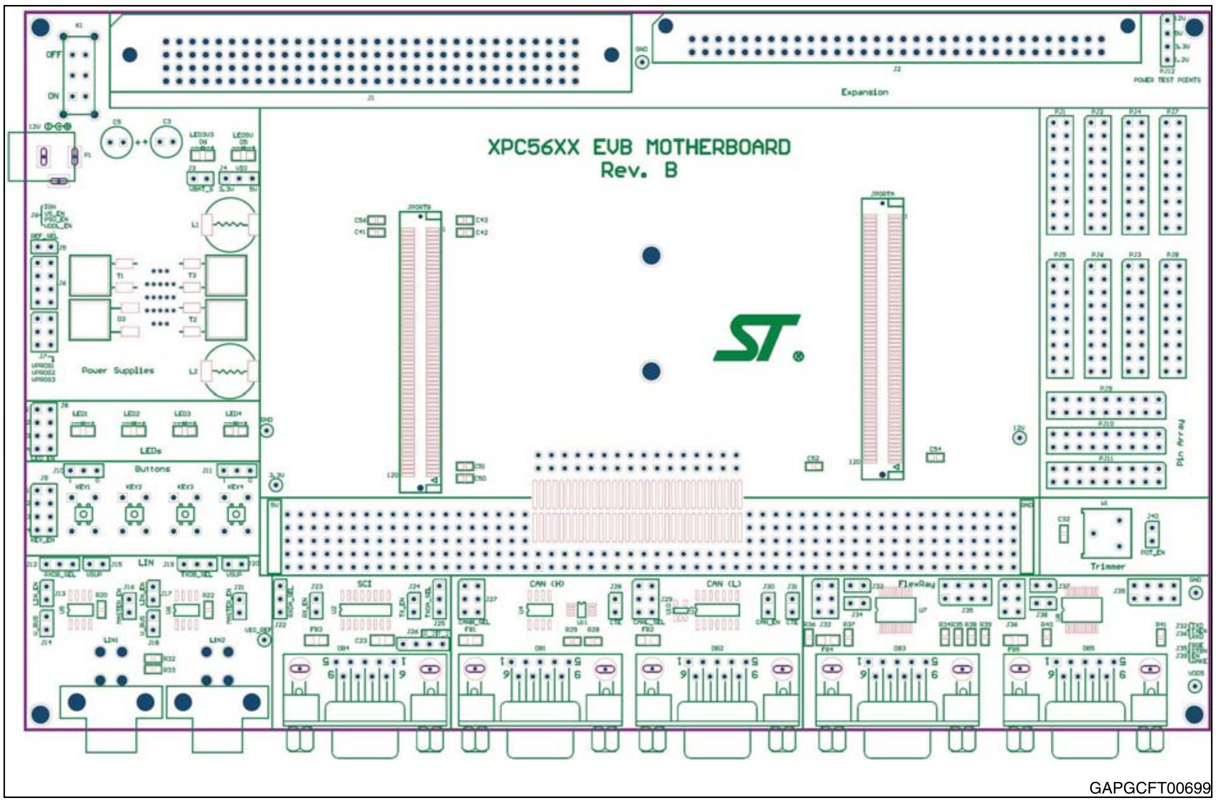
```
uint32_t FCCU_IRQ_Status(uint32_t CFG_TO_STAT, uint32_t* ALRM_STAT,
                            uint32_t* NMI_STAT){
    uint32_t exit_value= 0; /* Returned value = ERROR */

    if (CFG_TO_STAT == 1){
          FCCU.IRQ_STAT.B.CFG_TO_STAT |= 1;/* Clear the Configuration
                                    Time Out Error */
    }else{
          FCCU.IRQ_STAT.B.CFG_TO_STAT &= ~(1);/* No effect on bit */
    }
    *ALRM_STAT = FCCU.IRQ_STAT.B.ALRM_STAT;/* Read Alarm Interrupt Status */
    *NMI_STAT = FCCU.IRQ_STAT.B.NMI_STAT;/* Read NMI Interrupt Status */
```

```
        exit_value= 1;
        return(exit_value);
    }
```

A.3 General purpose functions

Two examples have been developed to show the features of FCCU. The examples have been implemented on the XPC56xxMB. The XPC56xxMB was plugged to XPC56EL mini-module with SPC56ELX 144 pins.

Figure 7. XPC56xxMB mother board



A.3.1 Example N1: fake NCF by external IRQ

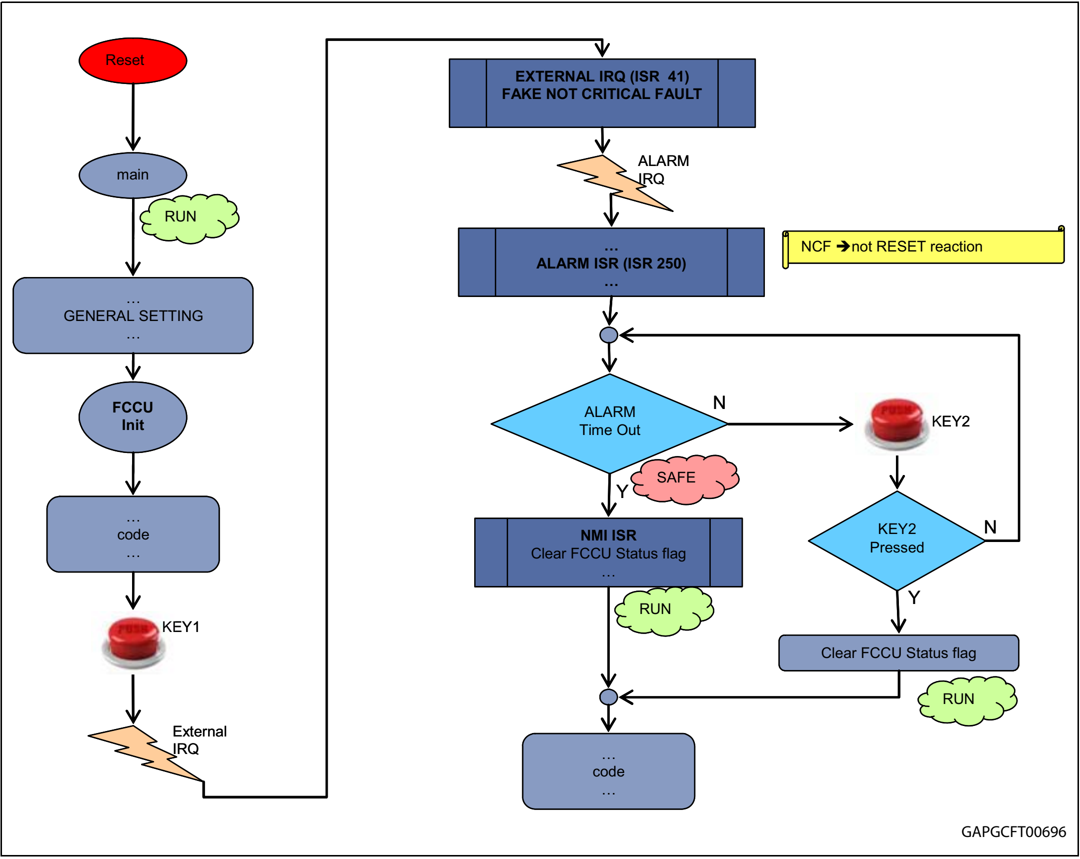
In this example we show the NCF fault injection (by external IRQ funtionality), in order to show an FCCU reaction. The example includes ALARM and NMI ISR assertion. The fault is checked and cleared with a subroutine by looking in the NCFSx registers.

Two external buttons have been used to inject and clear the NCF (KEY 1 and KEY 2) and LED1 on the motherboard has been used to view the fault status. To connect the two buttons to input pins (interrupt), we need to connect J8 pin 1B to JP9 pin 1, and J8 pin 2B to JP9 pin 2.

The flow is:

- Press key1: to inject asynchronous external IRQ (ISR)
- External IRQ (ISR)
  - Blinking LED1
  - Fake NCF
  - FCCU STATE = ALARM -> ALARM IRQ (ISR)
- Alarm IRQ (ISR)
  - Wait for ALARM-time-out (5 s) or check External push button (KEY2)
- If alarm-timeout
  - FCCU STATE = FAULT -> NMI IRQ (without RESET):
    - DEVICE STATE = SAFE
    - LED1 off
    - Clear FCCU FAULT and return to main
- If KEY 2 pressed
  - LED1 off
  - Clear FCCU FAULT and return to main

Figure 8. NCF injection flow



A.3.2 Example N2: fake CF by external IRQ

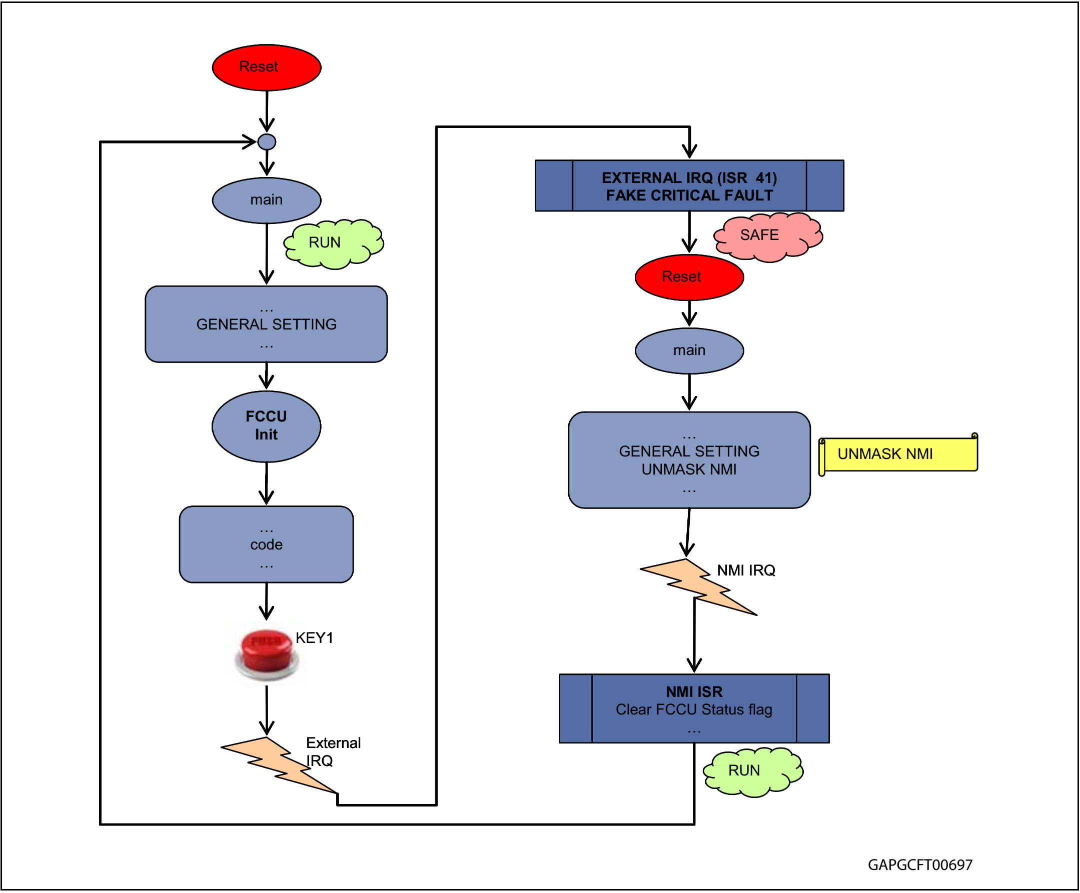
In this example we show the CF fault injection (by external IRQ funtionality), in order to show an FCCU reaction. The example includes NMI ISR assertion. The fault is checked and cleared with subroutine by looking in the CFSx registers.

This example employs the CF external button (KEY 1) to inject and clear, and the LED1 on the motherboard to view the fault. To connect the button to input pins (interrupt), we need to connect J8 pin 1B to JP9 pin 1.

The flow is:

- Blink LED1
- Press KEY1: to inject asynchronous external IRQ (ISR)
- External IRQ (ISR)
  - Fake CF
  - LED1 off
  - FCCU STATE = FAULT
  - DEVICE STATE = SAFE
  - RESET
  - NMI IRQ (ISR)
- After RESET: unmask NMI
- NMI IRQ (ISR)
  - Clear FCCU FAULT and return to main

Figure 9. CF injection flow



Appendix B Further information

B.1 Acronyms

Table 4. Acronyms

Acronym	Name
CRC	Cyclic redundancy check
DMA	Direct memory access
FCCU	Fault control and collection unit
INTC	Interrupt controller
MCU	Microcontroller unit
PIT	Periodic interrupt timer
TCD	Transfer control descriptor

Revision history

Table 5. Document revision history

Date	Revision	Changes
02-Aug-2012	1	Initial release.
17-Sep-2013	2	Updated disclaimer.