# SW drivers for the FS65xx family

**Table of Contents**

# Chapter 1: Introduction

This document describes the contents and the usage of the SW drivers for the FS65xx family. These drivers have been created for evaluation and demonstration purposes, allowing to rapidly configure the FS65xx device.

The drivers (.c, .h files) come with an example of project created with S32 Design Studio for Power Architecture for the MPC574xP target.

The FS65xx driver is MCU-independent, so the example project can be easily ported to another development tool or to another target.

# Chapter 2: SW package description

The package containing the drivers has the following directory structure:

| | |
|---|---|
| Doc | This folder contains this user guide |
| Drivers | This folder contains the MCU-independent driver files for the FS65xx family. These driver files consist in 3 files:<br><FS65xx.c> implements the driver c-function (public and private) and some user configurations that are described in the Chapter 3: FS65 SW driver description<br><FS65xx.h>  defines the FS65's register contents and addresses<br><FS65xx_drv.h> defines some internal structures used by the driver |
| Example | This folder contains the S32DS project for MPC5744P, illustrating the usage of the drivers. |

# Chapter 3: FS65 SW driver description

The FS65 driver works on top of a SPI driver that is *not* part of the driver since it is MCU-specific. A SPI driver for the MPC57xx family of MCU is provided in the example project. Please refer to the files DSPI.c and DSPI.h in the “Example” directory for a description of the API expected for sending/receiving SPI frames.

## 3.1 Public structures

The structures used by the FS65 drivers are defined in the FS65xx_drv.h and in FS65xx.c files. These public structures that can be used by the users are the following:

| Structure type | Description |
|---|---|
| `FS65_Registers_InitValues` | Initial values to be used for the initialization of the device when in INIT mode. This structure has a field for every INIT register of the FS65, and a field for the CAN_LIN_MODE, WD_WINDOW and LFSR registers. This structure is defined in <FS65xx.c>. The users should fill-in this structure according to their needs, in this file. |
| `ADCstruct` | This structure defines in the field “scanVoltage” which analog voltages available on the analog multiplexer output should be converted. This field “scanVoltage” has one bit for each analog voltages (Vref, Vsns, IO_0 etc…). A “1” indicates that the analog voltage should be converted (ie. output on the analog multiplexer of the FS65xx). The field <actualVoltage> records the last converted values for the ADC voltage, in the form of a float number. The users should define which analog voltages have to be converted, using the field “scanVoltage” of this structure. |
| `INTstruct` | This structure is defined in the file <FS65xx_drv.h>. It has one field for each register of the FS65. Each time a FS65 register is read, its value is recorded in this structure. |

## 3.2 Public Functions

### *3.2.1 Initialization functions*

| uint32_t FS65_Init_FSSM(void) | | |
|---|---|---|
| This function initializes all the INIT registers of the Fail-Safe State Machine according to the values stored in the structure "FS65_Registers_InitValues". In addition, the registers WD_WINDOW and WD_LFSR are also initialized | | |
| Return values | FS65_RETURN_OK | Operation successful |
| | Others | Number of errors encountered |

| uint32_t FS65_Init_MSM(void) | | |
|---|---|---|
| This function initializes all the INIT registers of the Main State Machine (excepted INIT_INT) according to the values stored in the structure "FS65_Registers_InitValues". INIT_INT should be initialized independently because after its initialization the INIT mode is exited. | | |
| Return values | FS65_RETURN_OK | Operation successful |
| | Others | Number of errors encountered |

| uint32_t FS65_Config_NonInit(void) | | |
|---|---|---|
| This function initializes some NORMAL registers of the FSSM and MSM. The register CAN_LIN_MODE is updated and the function FS65_UserConfigNonInit() is called. The user should add its own configuration in the function FS65_UserConfigNonInit(). The initial value is recorded in the structure "FS65_Registers_InitValues". | | |
| Return values | FS65_RETURN_OK | Operation successful |
| | Others | Number of errors encountered |

| void FS65_Init(void) |
|---|
| This function implements the full initialization of the FS65 device. It calls the function FS65_Init_FSSM() and FS65_Init_MSM(). When an error is encountered, the function FS65_ErrorCallback is called. The user should define this callback function. |

| void FS65_GetStatus(void) |
|---|
| This function reads the contents of the main registers of the FS65 device (diagnostic registers, wake-up sources, Device_ID etc…) and updates the INTstruct accordingly. Please refer to the code for a list of registers being accessed by this function. When an error is detected when reading a register, the user callback “FS65_ErrorCallback()” is called. |

### 3.2.2 User callbacks

The following functions are called by the FS65 drivers, and should be defined – if necessary – by the users.

| void FS65_ErrorCallback(void) |
|---|
| When an error is detected by the functions FS65_Init() or FS65_GetStatus(), this function is called. In the default implementation of this callback, the port PA1 is toggled every 1second. The users can change this behavior according to its application. |

| void FS65_UserConfig_NonInit(void) |
|---|
| This callback is called by the function FS65_Config_NonInit(). The user can complete the configuration of the non INIT registers in this function. |

```
void FS65_SPI_INT_Callback (void)
void FS65_PHYWU_INT_Callback(void)
void FS65_AutoWU_INT_Callback(void)
void FS65_LDTWU_INT_Callback(void)
void FS65_CAN_INT_Callback(void)
void FS65_VCORE_INT_Callback(void)
void FS65_VPRE_INT_Callback(void)
void FS65_VXXX_INT_Callback(void)
void FS65_IO_INT_Callback(void)
```

These callbacks are called by the interrupt routine FS65_IsrSIUL() which is called when a INTb pulse is generated by the FS65. Depending on the cause of the INTb pulse, different callbacks are called. These callbacks are respectively called for the following events:
SPI error
Wake-up from CAN or LIN
Auto Wake-up
Long Duration Timer wake-up
CAN or LIN event
Vcore events
Vpre events
Other regulator events
Event on IO

### 3.2.3 Interrupt routines

| FS65_IsrPIT_WD |
|---|
| This interrupt routine is attached to a periodic counter. It refreshes the FS65's watchdog: it reads the register WD_LFSR and then computes the new WD_ANSWER value and sends it. It automatically releases FS0b and optionally FS1b following a wake-up or loss of supply, after the Fault counter has been cleared (thanks to the watchdog refresh counter). |

| FS65_IsrSIUL |
|---|
| This interrupt routine is attached to the external interrupt connected to the INTb signal of the FS65xx. This routine handles all the events that could generate an interrupt pulse. It clears the corresponding flag and then call the user callback function corresponding to the event. See User Callbacks |

| FS65_IsrADC |
|---|
| This interrupt routine is attached to the End Of Conversion interrupt of the ADC module.This routine updates the structure ADCstruct with the converted values for the analog volatges of the AMUX output. |

### *3.2.4 Functions to configure the Long Duration Timer*

| uint32_t FS65_SelectLDTOperation(uint32_t) | |
|---|---|
| The function FS65_SelectLDTOperation selects the operating function of the Long Duration Timer | |
| Argument | Function should be either “LDT_FUNCTION_1”, “LDT_FUNCTION_2”, “LDT_FUNCTION_3”, “LDT_FUNCTION_4” or “LDT_FUNCTION_5” (these constants are defined in <FS65xx_drivers.h> |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_SetWakeUpReg (void) | |
|---|---|
| The function FS65_SetWakeUpReg configure the 3 registers LDT_WAKE_UP_x to read the 24bit Wake-up value and not the counter value | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_SetRTCReg (void) | |
|---|---|
| The function FS65_SetWakeUpReg configure the 3 registers LDT_WAKE_UP_x to read the current 24bit counter value Wake-up value and not the wake-up value. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_SetLDTNormalMode(void) | |
|---|---|
| The function FS65_SetLDTNormalMode selects the normal mode of operation for the LDT. Resolutions is 1sec | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `uint32_t FS65_SetLDTCalibrationMode(void)` | |
|---|---|
| `The function FS65_SetLDTCalibrationMode selects the calibration mode of operation for the LDT. Resolutions is 488µsec` | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `uint32_t FS65_StopLDTCounter(void)` | |
|---|---|
| `The function FS65_StopLDTCounter stops the LDT` | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `uint32_t FS65_StartLDTCounter(void)` | |
|---|---|
| `The function FS65_StopLDTCounter starts the LDT` | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `FS65_ConfAfterRunValue(uint32_t value)` | |
|---|---|
| `The function FS65_ConfAfterRunValue initializes the 2 registers LDT_AFTER_RUN_1 & LDT_AFTER_RUN_2 with the 16bit value given in argument` | |
| Argument | 16bit value. The higher byte goes in LDT_AFTER_RUN_1, the lower goes to LDT_AFTER_RUN_2 |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `FS65_ConfWakeUpValue(uint32_t value)` | |
|---|---|
| `The function FS65_ConfWakeUpValue initializes the 3 registers LDT_WAKE_UP_1,_2 & _3 with the 24bit value given in argument` | |
| Argument | 24bit value. The higher byte goes in LDT_WAKE_UP_1, the middle byte goes to LDT_WAKE_UP_2, and the lower byte goes to LDT_WAKE_UP_3 |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

### 3.2.4 Functions to change the mode of operation (MODE register)

| `uint32_t FS65_SetLPOFFmode(void)` | |
|---|---|
| The function FS65_SetLPOFFmode switches the current mode of the FS65xx to the low power OFF mode. This function sets bit GO_LPOFF in the MODE register to switch FS65xx into the Low power Vreg OFF mode. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `uint32_t FS65_SetLPOFFmode_autoWU (void)` | |
|---|---|
| The function FS65_SetLPOFFmode_autoWU switches the current mode of the FS65xx to the low power OFF mode with automatic WU 1msec after. This function sets bit LP_OFF_AUTO_WU in the MODE register to switch FS65xx into the Low power Vreg OFF mode with auto WU. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `uint32_t FS65_RequestINT (void)` | |
|---|---|
| This function sets bit INT_request in the MODE register, which generates an interrupt. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `uint32_t FS65_EnableVKAM (void)` | |
|---|---|
| This function sets bit VKAM_EN in the MODE register, which enables the Vkam regulator | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| `uint32_t FS65_DisableVKAM (void)` | |
|---|---|
| This function clears the bit VKAM_EN in the MODE register, which disables the Vkam regulator | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

### 3.2.5 Functions to configure the regulators (REG_MODE register)

| uint32_t FS65_EnableVcore (void) | |
|---|---|
| This function sets bit Vcore_EN in the REG_MODE register, which enables the Vcore power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_DisableVcore (void) | |
|---|---|
| This function clears bit Vcore_EN in the Reg Mode register, which disables the Vcore power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function. If Vcore has been defined to be safety-critical (register INIT_VCORE_OVUV_IMPACT), an error is returned and the regulator is not disabled. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_EnableVcca (void) | |
|---|---|
| This function sets bit Vcca_EN in the REG_MODE register, which enables the Vcca power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_DisableVcca (void) | |
|---|---|
| This function clears bit Vcca_EN in the Reg Mode register, which disables the Vcca power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function. If Vcca has been defined to be safety-critical (register INIT_VCCA_OVUV_IMPACT), an error is returned and the regulator is not disabled. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_EnableVaux (void) | |
|---|---|
| This function sets bit Vaux_EN in the REG_MODE register, which enables the Vaux power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_DisableVaux (void) | |
|---|---|
| This function clears bit Vaux_EN in the Reg Mode register, which disables the Vaux power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function. If Vaux has been defined to be safety-critical (register INIT_VAUX_OVUV_IMPACT), an error is returned and the regulator is not disabled. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_EnableVcan (void) | |
|---|---|
| This function sets bit Vcan_EN in the REG_MODE register, which enables the Vcan power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_DisableVcan (void) | |
|---|---|
| This function clears bit Vcan_EN in the Reg Mode register, which disables the Vcan power supply. Note that the register REG_MODE should have been read previously with the FS65_UpdateRegister() function. | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

### 3.2.6 Functions to configure the IO pins and the AMUX output (IO_Out_AMUX register)

| uint32_t FS65_EnableOUT4 (void) | |
|---|---|
| This function sets bit IO_out_4_EN (register IO_out-AMUX),which enables output gate driver for IO_4. Note that the register IO_OUT_AMUX should have been read previously | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_SetOUT4 (void) | |
|---|---|
| This function sets bit IO_out_4 (register IO_out-AMUX), which sets IO_4 to logical high state. Note that the register IO_OUT_AMUX should have been read previously | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| uint32_t FS65_ClearOUT4 (void) | |
|---|---|
| This function clears bit IO_out_4 (register IO_out-AMUX), which sets IO_4 to logical low state. Note that the register IO_OUT_AMUX should have been read previously | |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

| Unint32_t FS65_SwitchAMUXchannel (uint32_t value) | |
|---|---|
| This function switches AMUX channel to the desired configuration (bits Amux_2:0 in the IO_OUT_AMUX register. Note that the register IO_OUT_AMUX should have been read previously | |
| Argument | Number of the desired AMUX channel (from 0 till 7). Use the #defines defined in FS65_driver.h : AMUX_VREF, AMUX_VSNS_WIDE, AMUX_IO0_WIDE , AMUX_IO5_WIDE, AMUX_VSNS_TIGHT, AMUX_IO0_TIGHT, AMUX_IO5_TIGHT, AMUX_TEMP |
| Return Value | FS65_RETURN_OK - configured successfully<br>FS65_RETURN_ERROR - Operation failed or Wrong argument |

# Chapter 4 : Step-by-Step configuration of the FS65xx drivers

This chapter guides you through the successive steps required to configure the FS65xx drivers according to your needs.

1. **Define the version of your silicon in <FS65xx.h>**
   Only the second silicon version has been automotive-qualified and is mass-produced, so this is the default version supported by the FS65xx drivers. In case you still have the very first silicon version, you have to know that it has a slightly different register contents that should be taken into account: comment the line "`#define FS65_V11 1`" in <FS65xx.h>.
   If you are not sure about your silicon version, read the register DEVICE_ID. If the field DEV_REV is b0000, then you have the very first silicon.

2. **Define the initial values for the INIT registers**
   These initial values are defined in the structure "FS65_Registers_InitValues" in the file <FS65xx.c>. Each field of this structure corresponds to an INIT register, and holds the 16bit initialization value for this register.
   Some INIT registers have some secured bits. These secured bits will be automatically computed during the initialization, you can leave these bits at 0.

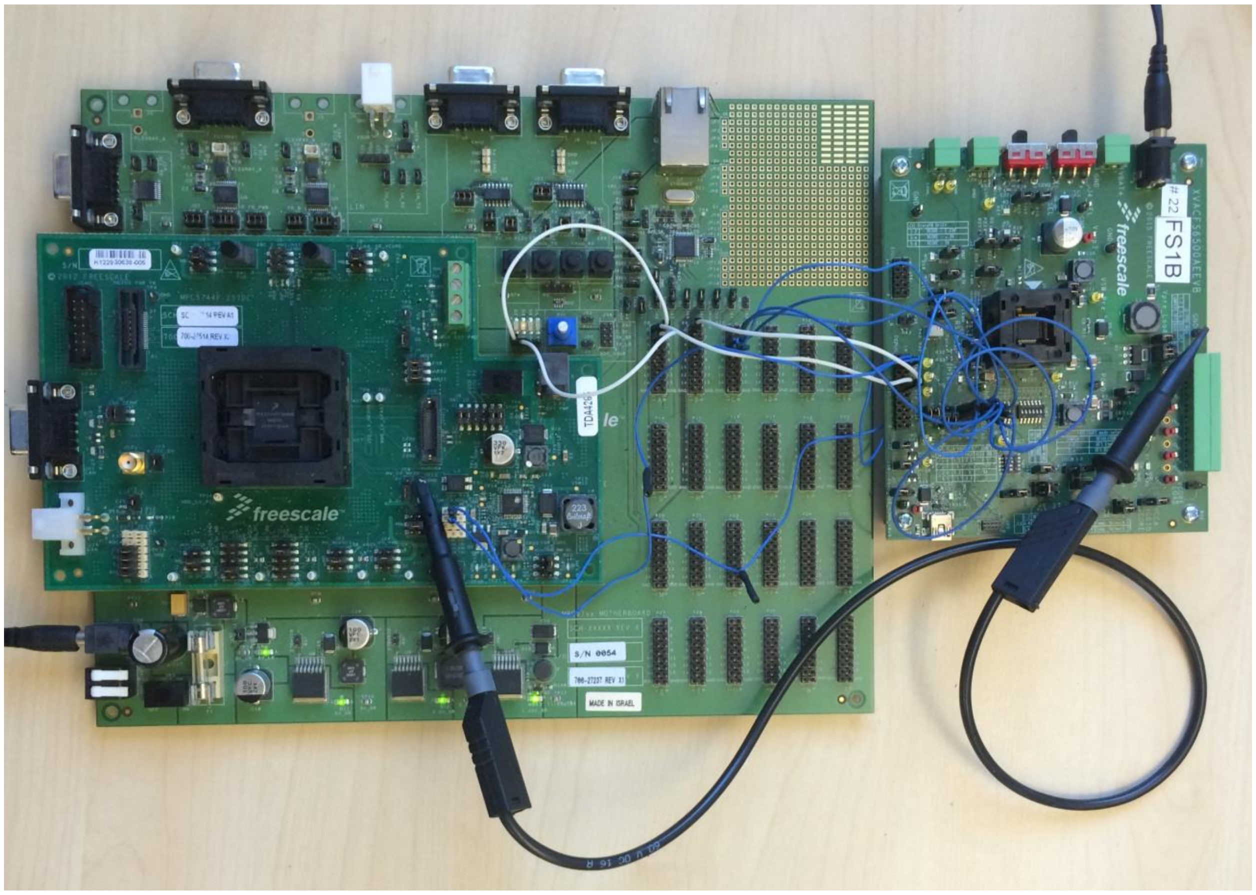
3. **If necessary, define the user callback functions, in the file <FS65xx.c>**
   Refer to the chapter §3.2.2 User callbacks" for a description of this callbacks.

# Chapter 5 : Set-up of the S32DS example project

This chapter describes how the example project works. This project has been developed for the MPC5744P. It has been tested with a MPC57xx Motherboard and a MPC5744P-257DC daughter board, connected to the FS65 evaluation board.

## 5.1 Hardware Set-up

The picture below shows the set-up used to develop the example of project.



## 5.2 Pin usage and connections

| Function | MCU port | MCU EVB pin | FS65 EVB pin |
|---|---|---|---|
| Interrupt signal between MCU and FS65xx | PA[0] | P8[1] | J37[12] |
| Debug pin 0 | PA[1] | P8[2] | NA |
| Debug pin 1 | PA[1] | P8[3] | NA |
| CAN Tx | PB[0] | J17[5] (daughter board) | J37[18] |
| CAN Rx | PB[1] | J17[2] (daughter board) | J37[19] |
| Analog multiplexer (MUX_OUT) | PB[7] | P9[1] | J37[11] |
| SPI SIN | PC[7] | P10[8] | J24[1] |
| SPI SOUT | PC[6] | P10[7] | J24[3] |
| SPI SCK | PC[5] | P10[6] | J24[7] |
| SPI CS | PC[4] | P10[5] | J24[5] |