# How to use STSW-DIGAFEV1DRV and a STM32 to control the TSC1641 through I2C

## Introduction

This technical note gives the basic information to utilize the STSW-DIGAFEV1DRV software library to control and use the TSC1641.

The TSC1641 is a high precision current, voltage, power, and temperature monitoring analog front-end (AFE).

It monitors current in a shunt resistor and load voltage up to 60 V in a synchronized way. The current measurement can be high-side, low-side, and bidirectional.

The device integrates a high precision 16-bit dual channel ADC with a programmable conversion time from 128 µs to 32.7 ms. Moreover, the TSC1641 allows the assertion of several alerts on the voltage, current, power, and temperature signals. Additionally, for best performance, the device's parameters and thresholds can be correctly set using the specific registers.

The TSC1641 supports I2C and MIPI I3C protocols.

This document describes how to communicate with the TSC1641 thanks to I2C.

# 1 Glossary

Table 1. **Glossary**

| Acronym | Description |
|---|---|
| I2C | Inter-integrated circuit. Two-wire communication protocol for connecting multiple devices, allowing data exchange and control within electronic systems |
| SCL | Clock line |
| SDA | Data line |
| NACK | NACK stands for "Not acknowledged" in I2C communication. It is a response from the receiver device indicating that it has not acknowledged the data sent to it by the controller device |
| Rshunt | Shunt resistor used for current sensing |
| STM32 | A family of 32-bit microcontrollers developed by STMicroelectronics. They are based on the Arm Cortex®-M processor architecture and are widely used in a variety of applications |

# 2 I2C configuration of your microcontroller

## 2.1 TSC1641 characteristics

The TSC1641 supports standard mode, fast mode, and fast mode plus (1 MHz).
The supported voltage is 3.3 V.
Up to four devices can be connected to the bus thanks to two address pins: A0 and A1.

**Table 2. TSC1641 characteristics**

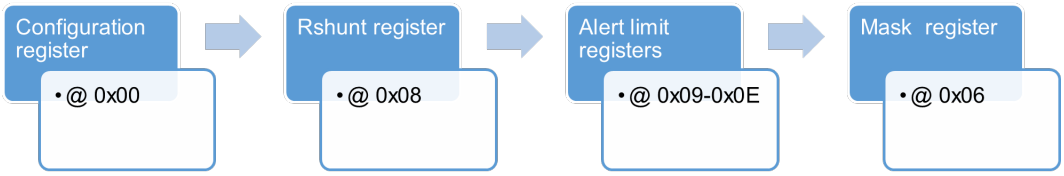
| A1 | A0 | Target address (binary) | Target address (hexa) |
|---|---|---|---|
| GND | GND | 10000000 | 40 |
| GND | VS | 10000001 | 41 |
| VS | GND | 10000010 | 42 |
| VS | VS | 10000011 | 43 |

For best performance, be sure to limit the bus capacitance if you use several I2C devices on the same bus.

# 3 Standard process of configuration

The standard process of configuration is the following:

**Figure 1. Process to configure the TSC1641**




## 3.1 Configuration register

The configuration register (@ 0x00(h)) permits to configure several characteristics of the device.
The configuration must be entered in a ***Configuration*** structure. This structure is located in the header file.

**Figure 2. Configuration structure**

```
81  // Definition of parameters in the configuration register :
82  typedef struct Configuration Configuration;
83  struct Configuration{
84  uint8_t TSC1641_RESET ;
85  uint8_t TSC1641_CT ;
86  uint8_t TSC1641_TEMP ;
87  uint8_t TSC1641_MODE ;
88  } ;
```

**Figure 3. Configuration register**

| Bit n° | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | RST | | | | | | | | CT3 | CT2 | CT1 | CT0 | TEMP | M2 | M1 | M0 |
| POR value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |

**Table 3. Links between variable and configuration register bits**

| Name | Bits of the configuration register |
|---|---|
| uint8_t TSC1641_RESET | Bit 15 |
| uint8_t TSC1641_CT | Bits 7-6-5-4 |
| uint8_t TSC1641_TEMP | Bit 3 |
| uint8_t TSC1641_MODE | Bits 2-1-0 |

Please refer to the datasheet or the application note AN5998 “Learning to use the TSC1641” to select the appropriate value to write in the registers.

**Figure 4. Several values to enter in the configuration register**

```
/*-----------------configuration register----------------------------*/
/*reset bit*/
#define TSC1641_rst_On              0x01  /*RST off*/
#define TSC1641_rst_Off             0x00  /*RST on */

/*conversion time*/
#define TSC1641_Conf_CT_128         0x00  /*conversion time of 128µs*/
#define TSC1641_Conf_CT_256         0x01
#define TSC1641_Conf_CT_512         0x02
#define TSC1641_Conf_CT_1024        0x03
#define TSC1641_Conf_CT_2048        0x04
#define TSC1641_Conf_CT_4096        0x05
#define TSC1641_Conf_CT_8192        0x06
#define TSC1641_Conf_CT_16384       0x07
#define TSC1641_Conf_CT_32768       0x08  /*conversion time of 32ms*/

/*temperature sensor*/
#define TSC1641_Temp_Off            0x00  /*Temperatrue sensor off*/
#define TSC1641_Temp_On             0x01  /*Temperatrue sensor on */

/*mode*/
#define TSC1641_Mode_ShutDown       0x00  /*mode shutdown                */
#define TSC1641_Mode_Vsh_Trig       0x01  /*mode trigger Vshunt only     */
#define TSC1641_Mode_Vload_Trig     0x02  /*mode trigger Vload only      */
#define TSC1641_Mode_Vshload_Trig   0x03  /*mode trigger Vshunt and Vload*/
#define TSC1641_Mode_Idle           0x04  /*mode Idle                    */
#define TSC1641_Mode_VshCont        0x05  /*mode continuous Vshunt only  */
#define TSC1641_Mode_VloadCont      0x06  /*mode continuous Vload only   */
#define TSC1641_Mode_VshloadCont    0x07  /*mode continuous Vshunt and Vload*/
```

The header file contains several values that help to configure the TSC1641.

Then, use the *TSC1641_SetConf(I2C_HandleTypeDef *hi2c1, Configuration * CONF1)* to write the desired value in the configuration register.

**Figure 5. Example of configuration**

```
/* Example of Configuration*/
Configuration config;
Configuration *pConfig = &config;
pConfig->TSC1641_RESET = TSC1641_rst_Off; /*reset bit to 0*/
pConfig->TSC1641_CT = TSC1641_Conf_CT_32768; /* conversion time of 32ms*/
pConfig->TSC1641_TEMP = TSC1641_Temp_On; /* Temperature sensor is on*/
pConfig->TSC1641_MODE = TSC1641_Mode_VshloadCont; /* Mode continuous Vshunt and Vload*/
TSC1641_SetConf(&hi2c1, pConfig); /* write the config in the configuration register*/
```

**Table 4. Configuration register (@ 0x00) filled by the TSC1641 SetConf() function**

| Bit n° | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | RST | | | | | | | | CT3 | CT2 | CT1 | CT0 | TEMP | M2 | M1 | M0 |
| Value written | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | **0** | **0** | **0** | **1** | **1** | **1** | **1** |

## 3.2 Rshunt register

The shunt resistor value is defined in the header file.

**Figure 6. Shunt resistor value**

```
/*------------------Shunt value-------------------------------------*/
// In this example, Rshunt LSB is 5µOhm so Rshunt of 5mOhm = 0d1000 0x01f4
#define TSC1641_RShunt_Val          0x01f4
```

The LSB is 5µΩ. Hence the value to enter is:

$$Value_{toWrite} = \frac{RShunt_{val}}{RShunt_{LSB}}$$

In this example, the shunt resistor value is 5mΩ.

$$Value_{toWrite} = \frac{5*10^{-3}}{5*10^{-6}} = 1000(\mathrm{d}) = 0\mathrm{x}01\mathrm{f}4\left(\mathrm{h}\right)$$

Once the value defined, the value has to be written in the Rshunt register (@ 0x08) with the *TSC1641_SetRShunt(I2C_HandleTypeDef *hi2c1)* function.

**Figure 7. Use of the SetRshunt function**

```
TSC1641_SetRShunt(&hi2c1);
```

## 3.3 Alert limit registers

The TSC1641 has several configurable alerts.

**Table 5. Table with all the limit registers**

| Name | Description | address |
|---|---|---|
| SOL alert limit register | Shunt voltage over limit | 09h |
| SUL alert limit register | Shunt voltage under limit | 0Ah |
| LOL alert limit register | Load voltage over limit | 0Bh |
| LUL alert limit register | Load voltage under limit | 0Ch |
| POL alert limit register | Power over limit | 0Dh |
| TOL alert limit register | Temperature over limit | 0Eh |

For more information regarding these alerts, please refer to the Datasheet or the application note AN5998 “Learning to use the TSC1641”.

In the header file all the registers are already defined to simplify the programming.

**Figure 8. Treshold registers**

```
#define TSC1641_RegAdd_VshuntOV     0x09            //Alert threshold. LSB=2.5µV
#define TSC1641_RegAdd_VshuntUV     0x0A            //Alert threshold. LSB=2.5µV
#define TSC1641_RegAdd_VloadOV      0x0B            //Alert threshold. LSB=2mV
#define TSC1641_RegAdd_VloadUV      0x0C            //Alert threshold. LSB=2mV
#define TSC1641_RegAdd_PowerOL      0x0D            //Alert threshold. LSB=2.5mW. Works only if VLOAD>0
#define TSC1641_RegAdd_TempOL       0x0E            //Alert threshold. LSB=0.5°C
```

**Table 6. Defined constants with register addresses**

| Name | Name in the header file | Address | Comments |
|---|---|---|---|
| SOL alert limit register | TSC1641_RegAdd_VshuntOV | 09h | LSB=2.5µV |
| SUL alert limit register | TSC1641_RegAdd_VshuntUV | 0Ah | LSB=2.5µV |
| LOL alert limit register | TSC1641_RegAdd_VloadOV | 0Bh | LSB=2mV |
| LUL alert limit register | TSC1641_RegAdd_VloadUV | 0Ch | LSB=2mV |
| POL alert limit register | TSC1641_RegAdd_PowerOL | 0Dh | LSB=25mW.<br>Works only if VLOAD>0 |
| TOL alert limit register | TSC1641_RegAdd_TempOL | 0Eh | LSB=0.5°C |

The limits are set in the **Limit** structure.

**Figure 9. limit structure**

```
120  // Definition of alert thresholds :
121  typedef struct Limit Limit;
122  struct Limit{
123  uint16_t VSHUNT_OV_LIM ;  // Vsunt Over voltage limit value
124  uint16_t VSHUNT_UV_LIM ;    // Vshunt Under voltage limit value
125  uint16_t VLOAD_OV_LIM ; // Vload Over voltage limit value
126  uint16_t VLOAD_UV_LIM ;   // Vload Under voltage limit value
127  uint16_t POWER_OV_LIM ;   // Power over limit value
128  uint16_t TEMP_OV_LIM ;    // Temperature over limit value
129  } ;
```

In the following example, all the thresholds are set with different values:

**Table 7. Values to write in the limit registers**

| Name | Name in the header file | Value (SI) | Value to write (hexadecimal) |
|---|---|---|---|
| SOL alert limit register | TSC1641_RegAdd_VshuntOV | 10 mv | 0x0FA0 |
| SUL alert limit register | TSC1641_RegAdd_VshuntUV | 5mv | 0x07D0 |
| LOL alert limit register | TSC1641_RegAdd_VloadOV | 30V | 0x3A98 |
| LUL alert limit register | TSC1641_RegAdd_VloadUV | 15V | 0x1D4C |
| POL alert limit register | TSC1641_RegAdd_PowerOL | 200W | 0x1F40 |
| TOL alert limit register | TSC1641_RegAdd_TempOL | 70°C | 0x008C |

**Figure 10. Values written in the limit structure**

```
// limit for each threshold
Limit limit;
Limit *pLimit = &limit;
pLimit->VSHUNT_OV_LIM = 0x0FA0;
pLimit->VSHUNT_UV_LIM = 0x07D0;
pLimit->VLOAD_OV_LIM = 0x3A98;
pLimit->VLOAD_UV_LIM = 0x1D4C;
pLimit->POWER_OV_LIM = 0x1F40;
pLimit->TEMP_OV_LIM = 0x008C;
TSC1641_SetLimits(&hi2c1, pLimit); // function to write in the limit registers
```

At the end, the *TSC1641_SetLimits(I2C_HandleTypeDef *hi2c1, Limit* LIMIT)* is called to write all the values in the several registers.

*Note: It is not mandatory to write in all the limit registers. It is possible to only write the limits to use.*

## 3.4 Mask register

The last step is to activate the desired alerts. To do this, bits has to be set to 1 in the Mask register (@0x06)
In the header file a Flag structure is described. It includes all the possible flags.

**Figure 11. Structure gathering the activable alerts in the TSC1641**

```
// Definition of the parameters in alert register :
typedef struct Flag Flag;
struct Flag{
uint8_t TSC1641_OVF ;    // Math Overflow Flag
uint8_t TSC1641_SATF ;   // Measurement saturation Flag
uint8_t TSC1641_SOF ;    // Shunt Voltage Over voltage
uint8_t TSC1641_SUF ;    // Shunt Voltage Under voltage
uint8_t TSC1641_LOF ;    // Load Voltage Over voltage
uint8_t TSC1641_LUF ;    // Load Voltage Under voltage
uint8_t TSC1641_POF ;    // Power Over Limit
uint8_t TSC1641_TOF ;    // Temperature Over Limit
uint8_t TSC1641_CVRF ;   // Conversion ready alert enable
} ;
```

In the following example, the alerts and parameters described in Table 8. Alerts activated or not for this example are activated by writing in the Mask register.

**Table 8. Alerts activated or not for this example**

| Variable | To activate |
|---|---|
| TSC1641_SOL | yes |
| TSC1641_SUL | no |
| TSC1641_LOL | yes |
| TSC1641_LUL | no |
| TSC1641_POL | yes |
| TSC1641_TOL | no |
| TSC1641_CVNR | yes |
| TSC1641_APOL | no |
| TSC1641_ALEN | yes |

The desired alerts and parameters are selected in a Flag structure, then this structure is given as a parameter for the *TSC1641_SetAlerts(I2C_HandleTypeDef *hi2c1, Alert* ALERT1)*. The way to use the structure and the function is described in Figure 12. Process to write the mask register.

**Figure 12. Process to write the mask register**

```
// Alerts to activate
Alert alert;
Alert *pAlert = &alert;
pAlert->TSC1641_SOL = TSC1641_Alert_On;
pAlert->TSC1641_SUL = TSC1641_Alert_Off;
pAlert->TSC1641_LOL = TSC1641_Alert_On;
pAlert->TSC1641_LUL = TSC1641_Alert_Off;
pAlert->TSC1641_POL = TSC1641_Alert_On;
pAlert->TSC1641_TOL = TSC1641_Alert_Off;
pAlert->TSC1641_CVNR = TSC1641_Alert_On;
pAlert->TSC1641_APOL = TSC1641_Alert_Off;
pAlert->TSC1641_ALEN = TSC1641_Alert_On;
TSC1641_SetAlerts(&hi2c1, pAlert); // write the value in the Mask register
```

At the end of the process, the Mask register is the following:

**Table 9. Mask register value**

| Bit n° | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | SOL | SUL | LOL | LUL | POL | TOL | CVNR | | | | | | | | APOL | ALEN |
| Value written | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

# 4 Collect of data

## 4.1 Read of sensed data

The STSW-DIGAFEV1DRV software has several functions to read the values sensed or computed by the TSC1641.

**Figure 13. Functions to read the sensed or calculated data**

```
137  void TSC1641_GetShuntVal(I2C_HandleTypeDef *hi2c1, uint8_t Data[]);
138  void TSC1641_GetVloadVal(I2C_HandleTypeDef *hi2c1, uint8_t Data[]);
139  void TSC1641_GetCurrentVal(I2C_HandleTypeDef *hi2c1, uint8_t Data[]);
140  void TSC1641_GetPowerVal(I2C_HandleTypeDef *hi2c1, uint8_t Data[]);
141  void TSC1641_GetTempVal(I2C_HandleTypeDef *hi2c1, uint8_t Data[]);
```

They work all in the same way. They read the adequate register (@ 0x01 to 0x05 according to the desired value) and store the data in a uint8_t table.

**Table 10. Register map of sensed/calculated data**

| Pointer address (hex) | Register name |
|---|---|
| 01h | Shunt voltage register |
| 02h | Load voltage register |
| 03h | DC power register |
| 04h | Current register |
| 05h | Temperature register |

For example, these two lines are the only needed to read the shunt value and the shunt voltage is stored in the two bytes variable Data_shunt:

**Figure 14. Shunt value example**

```
uint8_t Data_shunt[2];
TSC1641_GetShuntVal(&hi2c1,Data_shunt);
```

## 4.2 Get the alerts

As mentioned in the Section 3.4: Mask register, the TSC1641 has several activable alerts.

### 4.2.1 Read the flag register

The alert states, the saturation flag and the math overflow flag are accessible by reading the Flag register (@0x07).

**Figure 15. Function to read the flag register**

```
TSC1641_GetAlert(&hi2c1, pFlag);
```

The state of each flag is stored in a flag structure.

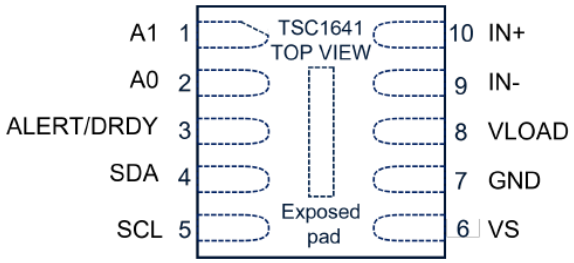
Figure 16. **Flag structure**

```
105  // Definition of the parameters in alert register :
106  typedef struct Flag Flag;
107  struct Flag{
108  uint8_t TSC1641_OVF ;    // Math Overflow Flag
109  uint8_t TSC1641_SATF ;   // Measurement saturation Flag
110  uint8_t TSC1641_SOF ;    // Shunt Voltage Over voltage
111  uint8_t TSC1641_SUF ;    // Shunt Voltage Under voltage
112  uint8_t TSC1641_LOF ;    // Load Voltage Over voltage
113  uint8_t TSC1641_LUF ;    // Load Voltage Under voltage
114  uint8_t TSC1641_POF ;    // Power Over Limit
115  uint8_t TSC1641_TOF ;    // Temperature Over Limit
116  uint8_t TSC1641_CVRF ;   // Conversion ready alert enable
117  } ;
118
```

### 4.2.2 Be informed via the pin alert

As in the figure below, an ALERT/Data ReaDY pin is accessible on pin 3:

Figure 17. **TSC1641 pinout**




According to the chosen polarity defined in the Alert structure, the TSC1641 ALERT/DRDY will be to the one logic or the zero logic when a flag is risen.

To detect the rising flag, the STM32 can use an interruption.

Figure 18. **Example of an interrupt**

```
344  /* USER CODE BEGIN 4 */
345  /* Interruption */
346  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
347  {
348    HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
349  }
350
```

In this example the LED 2 is toggled when an alert is triggered.

# Revision history

Table 11. **Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 16-Feb-2024 | 1 | Initial release. |

# Contents

# List of tables

# List of figures