README

This document provides instructions on how to use the driver and precautions.

1. Configuration

```
#define OW_DISABLE_IRQ()                __disable_irq();
#define OW_ENABLE_IRQ()                 __enable_irq();
#define OW_NO_RESPOND_THRESHOLD_5msTick     (200u)
#define OW_RDYCHK_EXTRA_5msTick         (2)
#define OW_CMDQ_SIZE                    (30u)

#define TEMPSENS_9BIT_RESOL_WAIT_5msTick    (105u)
#define TEMPSENS_10BIT_RESOL_WAIT_5msTick   (105u)
#define TEMPSENS_11BIT_RESOL_WAIT_5msTick   (105u)
#define TEMPSENS_12BIT_RESOL_WAIT_5msTick   (105u)
#define TEMPSENS_DATA_INVALID_CNT_THRSLD    (5)
```

Name	Description	Default
`OW_DISABLE_IRQ`	IRQ disable macro used in the 1-Wire layer	`__disable_irq()`
`OW_ENABLE_IRQ`	IRQ enable macro used in the 1-Wire layer	`__enable_irq()`
`OW_NO_RESPOND_THRESHOLD_5msTick`	Maximum allowed duration to wait for sensor response	`200u` (1sec)
`OW_RDYCHK_EXTRA_5msTick`	Additional margin added to temperature conversion wait time	`2` (10ms)
`OW_CMDQ_SIZE`	Command queue size for the 1-Wire layer	`30` u
`TEMPSENS_DATA_INVALID_CNT_THRSLD`	Maximum allowed consecutive CRC mismatches	`5`
`TEMPSENS_9BIT_RESOL_WAIT_5msTick`	Conversion wait time for 9-bit resolution	`105u` (510ms)

`TEMPSENS_10BIT_RESOL_WAIT_5msTick`	Conversion wait time for 10-bit resolution	`105u` (510ms)
`TEMPSENS_11BIT_RESOL_WAIT_5msTick`	Conversion wait time for 11-bit resolution	`105u` (510ms)
`TEMPSENS_12BIT_RESOL_WAIT_5msTick`	Conversion wait time for 12-bit resolution	`105u` (510ms)

⚠️ Caution !

1. **This driver is designed for a *single sensor per MCU* configuration and assumes an *external power supply mode*.**

   Do not use this driver outside the intended scope.

2. `RESOL_WAIT_5msTick` **values may vary depending on the sensor characteristics.**

   If frequent timeout errors occur during temperature conversion, adjust the `RESOL_WAIT_5msTick` values accordingly.

3. **Set** `OW_CMDQ_SIZE` **to at least 20 entries.**

   A single temperature read operation requires 17 individual 1-Wire commands.

2. Initialize

Before starting the scheduler, call the following functions in `main()`

- `oneWire_stMachine_init`
- `tempSens_stMachine_init`

⚠️ The 1-Wire initialization function must be called first.

example

```
    HAL_TIM_Base_Start(&htim2); // Timer for 1-Wire microsecond delay
    oneWire_stMachine_init(TEMP_SENSOR_DIO_GPIO_Port, TEMP_SENSOR
_DIO_Pin, &htim2);
    tempSens_stMachine_init();
```

3. Require Func Cmd

```
bool result_readTemp_flag = false;
bool failed_readTemp_flag = false;
bool available_flag = false;
typTempSens_errCode tempSens_errcode = TEMPSENS_ERR_OKAY;

void forTest_500ms(void)
{
   tempSens_available_flag = tempSens_isReady_forReq();

   if(tempSens_available_flag == true)
   {
      result_readTemp_flag = tempSensor_reqCommand(TEMPSENS_CMD_
REQ_DATA,
                                                   NULL, &failed_readTemp_flag);

      fnd_setFloat(tempSensor_getTemper_celcius(),CENTI_RESOL);
   }
   else
   {
      tempSens_errcode = tempSensor_getErrCode();
   }
}
```

Usage Flow

1. Call `tempSens_isReady_forReq()` to check whether the driver is ready to accept a new request.
2. If the function returns `true`, send a request using `tempSensor_reqCommand()`.
3. If it returns `false`, retrieve the error reason using `tempSensor_getErrCode()`.
4. If no error is reported, the sensor may currently be:
    a. detecting an error condition
    b. waiting for temperature conversion to complete.

If `failed_readTemp_flag` becomes **true**, the request was **not successfully forwarded to the lower layer**.

In this case, retry the request after some time.

If the issue persists, inspect the state of each layer using a debugger.