How to retain RAM data in reset procedure and low-power mode transition in TRAVEO™ family

About this document

Scope and purpose

This application note explains the procedure to ensure RAM retention in TRAVEO™ T2G family MCU when a software reset or low-power mode transition occurs.

Intended audience

This document is intended for anyone using TRAVEO™ T2G family.

Table of contents

1 Introduction

This application note describes the procedure for reset and low-power mode transitions in Infineon TRAVEO™ T2G family CYT2/CYT3/CYT4 series devices to ensure RAM retention.

In these devices, a reset is performed asynchronously regardless of the RAM access status. Therefore, if a reset occurs during operation, the RAM data might be lost. In addition, if the device power mode transitions from active to low power, you should follow the appropriate procedure for RAM retention. This document describes the procedures to ensure RAM data retention after a software reset or low-power mode transition. However, in Hibernate mode, the RAM data cannot be retained. Therefore, the RAM data should be transferred to the application flash once. After returning to Active mode, it is necessary to return the RAM data from the application flash to RAM. In this case, the transition data is defined as backup memory data.

To understand the functionality described and terminology used in this application note, see the "SRAM Interface" and "Work Flash" chapter of the architecture technical reference manual (TRM).

2 RAM retention procedure overview

2.1 Reset procedure

Figure 1 shows the flow of RAM retention when a reset occurs. This example shows the case of retaining RAM0 data.

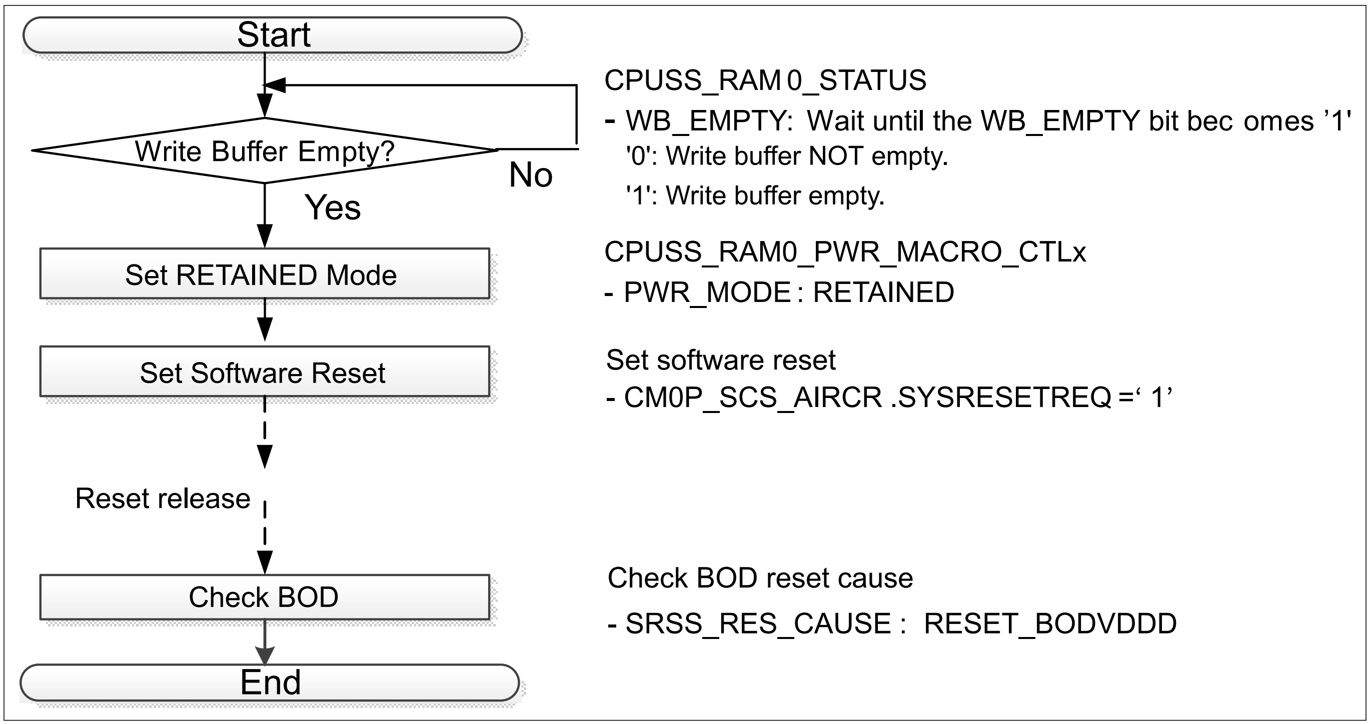

Figure 1 **Example of RAM0 retention procedure**

First, in the case of RAM0, check the write buffer status of the WB_EMPTY bit of the CPUSS_RAM0_STATUS register. The WB_EMPTY bit indicates whether there is any data in the write buffer or it is empty.

In CYT2 series MCU, ECC is added to the 32-bit data. Therefore, if a partial AHB-Lite write (8-bit/16-bit) occurs to the RAM, the missing data is read from the RAM. Then, the missing data and partial write data are merged to generate the 32-bit complete data. ECC is calculated over the 32-bit complete data, and written to the RAM with the 32-bit data.

The CYT3/CYT4 series MCUs have the AXI bus interface. ECC of the AXI bus interface is added to the 64-bit data. Therefore, when a partial write (8-bit/16-bit/32-bit) occurs to the RAM, the missing data is read from the RAM. Then, the missing data and partial write data are merged to generate the 64-bit complete data. ECC is calculated over the 64-bit complete data.

The write buffer is used in this operation. Therefore, there is a possibility that write buffer has data that is not yet written to the RAM. To prevent missing of unwritten data in the write buffer, you should check the status of the write buffer.

If there is valid data, wait until it is written to RAM0. When there is no valid data in the write buffer, set the PWR_MODE bit of the CPUSS_RAM0_PWR_MACRO_CTLx register to RETAINED mode. Finally, generate a software reset. This procedure retains the RAM0 data and executes a reset. However, note that the RAM0 data cannot be retained if the voltage is lower than the brown-out detection (BOD: 2.7 V) level. Therefore, you should confirm that BOD has not occurred after returning from a reset.

Table 1 shows the RAM0 status register. You should confirm that the WB_EMPTY bit is set to '1' before initiating a software reset.

Table 1 RAM0 Status register

Register	Bit field	Bit value	Description
CPUSS_RAM0_STATUS	WB_EMPTY [0]	0	Write buffer not empty
		1	Write buffer empty

Table 2 shows the Power Control register, which controls the system RAM0 power states with a single macro.

Table 2 Power Control register

Register	Bit field	Bit value	Description
CPUSS_RAM0_PWR_MACRO_CTLx	PWR_MODE [1:0][1)]	0	OFF mode: Turns OFF the SRAM. This turns OFF both array and periphery power of the SRAM; SRAM memory contents are lost.
		1	Reserved
		2	RETAINED mode: Keeps the SRAM in Retained mode. This will turn OFF the SRAM periphery power, but the array power is ON to retain memory contents. SRAM contents will be retained in DeepSleep system power mode.
		3	ENABLE mode: Enables the SRAM for regular operation. SRAM contents will be retained in DeepSleep system power mode. (Default)

This register is for the CPUSS system's RAM0 controller. This information is used when the RAM0 RETAINED mode is set.

2.2 Low-power mode (DeepSleep mode) transition procedure

Figure 2 shows the flow of RAM retention for low-power mode transition. In this method, when transitioning to a low-power mode, settings for RAM retention are performed. The procedure to check the status of the write buffer and the RETAINED mode setting of RAM is the same. When the MCU enters a low-power mode, the main program execution stops. If a wakeup interrupt is generated, the MCU returns to Active mode.

[1] To set the PWR_MODE bit field, use word access in the CPUSS_RAM0_PWR_MACRO_CTLx register. See the registers TRM for details.

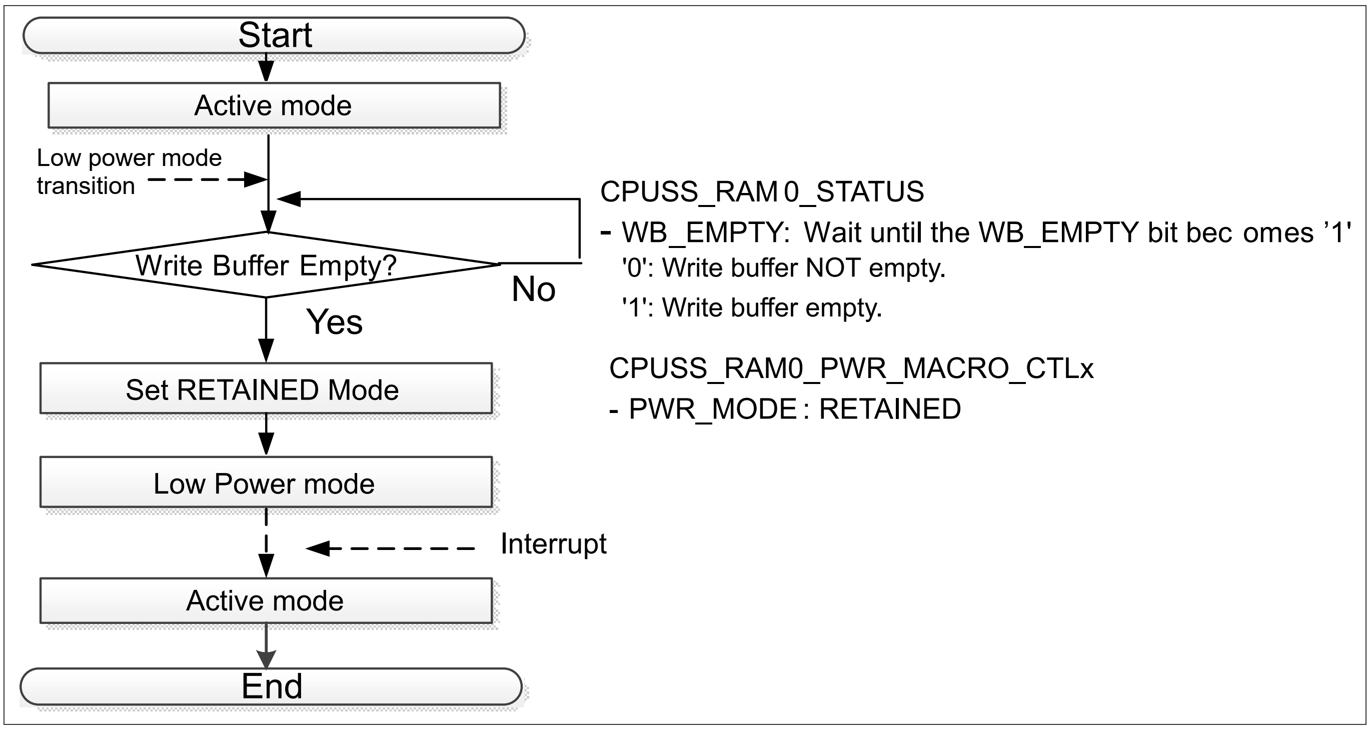

Figure 2 **Example of RAM0 retention procedure for low-power mode**

2.3 Low-power mode (Hibernate mode) transition procedure

Figure 3 shows how the backup memory data is backed up for Hibernate mode transition. In this method, when transitioning to Hibernate mode, the backup memory data of the RAM is transferred to the application flash. When the MCU enters Hibernate mode, the main program execution stops. If a Hibernate wakeup reset is generated, the MCU returns to Active mode, and the backup memory data is transferred to the RAM from the application flash. The transfer of data from the RAM to the application flash and back are handled by the user software.

To prevent unintentional overwriting of the backup data, you should consider not allowing other programs to access the backup data area of the application flash and RAM during data backup. In addition, you should ensure that the data transition time between the RAM and application flash meets your system requirements.

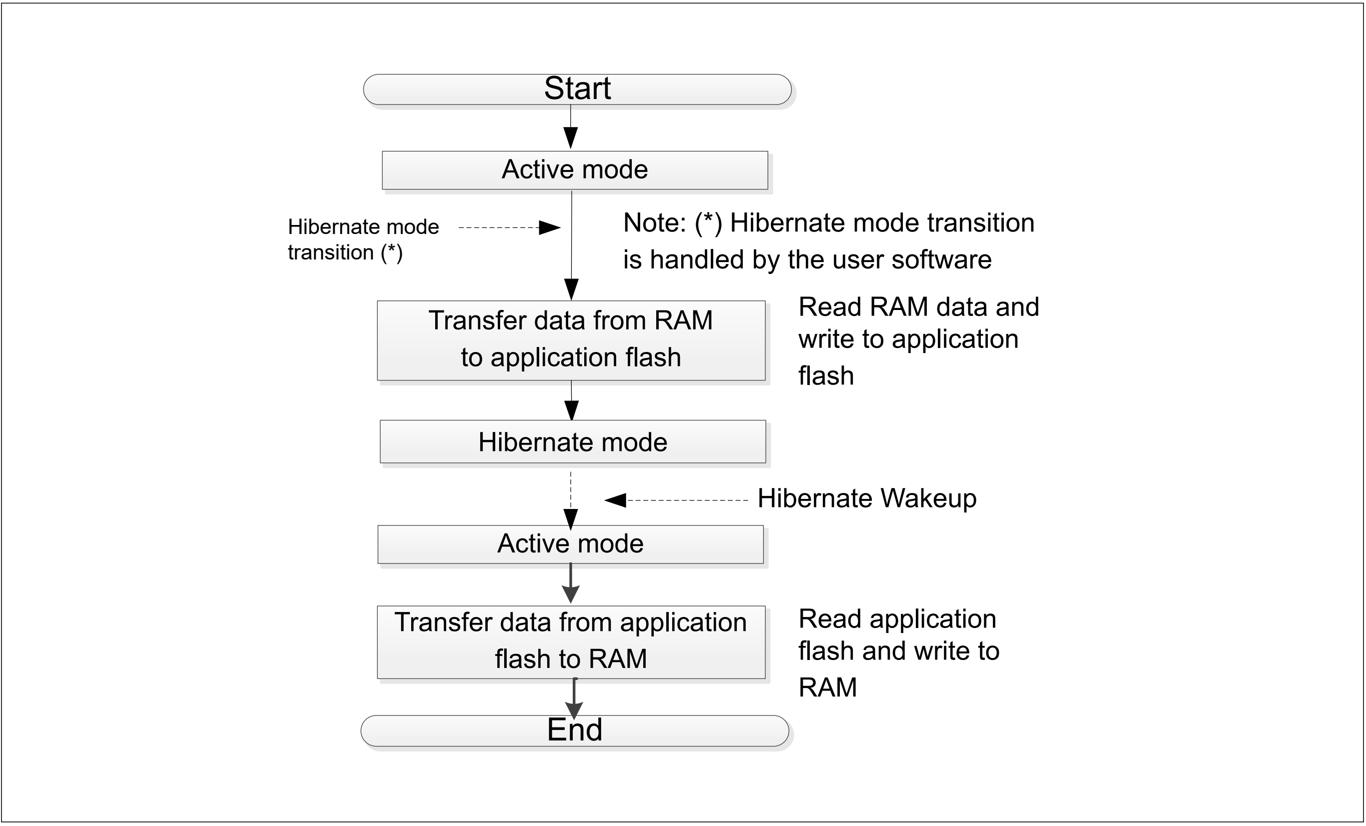
Start
Active mode
Hibernate mode transition (*)
Note: (*) Hibernate mode transition is handled by the user software
Transfer data from RAM to application flash
Read RAM data and write to application flash
Hibernate mode
Hibernate Wakeup
Active mode
Transfer data from application flash to RAM
Read application flash and write to RAM
End

Figure 3 **Example of backup procedure for Hibernate mode**

3 RAM retention procedure in reset

This section shows two examples for reset procedure with a block diagram, timing chart, and flowchart. One method uses the low-voltage detection (LVD) interrupt. The other method uses the external reset input signal of the external LVD IC.

3.1 Reset using LVD interrupt

In this case, LVD1 and LVD2 are used. LVD1 is a system low-level voltage detector that ensures that a reset occurs with guaranteed RAM0 retention. Also, LVD1 is used to detect a VDDD supply voltage drop. In addition, the user application starts by checking LVD2. LVD2 is used to check whether the VDDD supply voltage has recovered by setting the rising trip point.

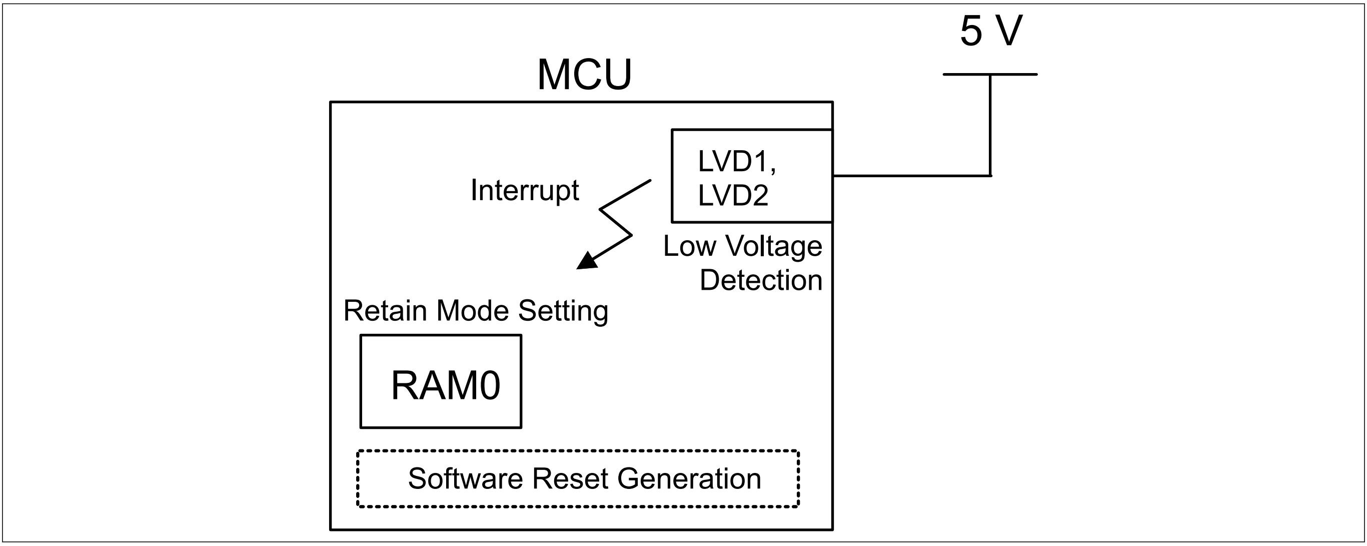

Figure 4 Block diagram for reset procedure for RAM0 retention with the LVD interrupt method

In Figure 4, first, an interrupt is generated when a falling edge is detected from LVD1. After the interrupt has occurred, RAM0 status must be checked and set to RETAINED mode. After these steps, perform a software reset. Then, after the MCU starts up from reset, check for VDDD exceeding the LVD2 rising edge with the SRSS_INTR register. Finally, confirm that a BOD reset has not occurred. If a BOD reset occurred, RAM0 data retention is not guaranteed. Therefore, RAM0 data must be discarded.

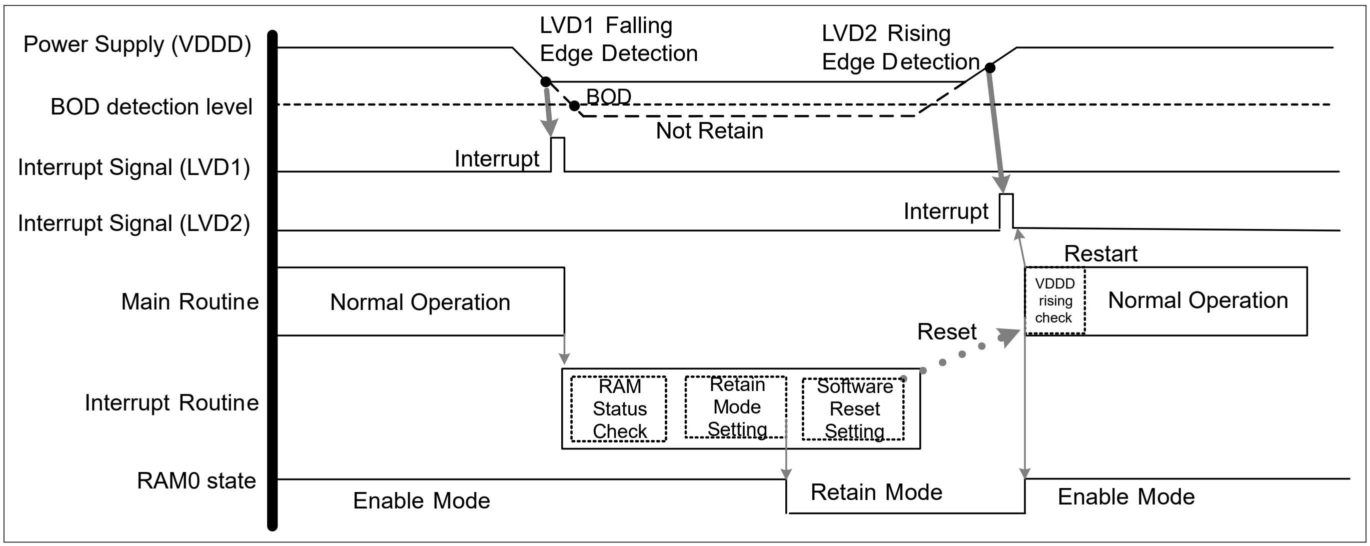

Figure 5 Example of reset procedure timing chart in RAM0 by LVD interrupt

In Figure 5, when the VDDD drops, the interrupt routine is called. Then, the interrupt routine checks the RAM0 status, RETAINED mode setting, and software reset setting (reset generation). After the reset is completed, ensure that VDDD is rising with the LVD2 detection method, and resume normal operation.

3.1.1 Use case

This section explains how to configure the reset using LVD interrupt based on a use case using the Sample Driver Library (SDL). The code snippets in this application note are part of SDL. See Other references for the SDL.

SDL has a configuration part and a driver part. The configuration part mainly configures the parameter values for the desired operation. The driver part configures each register based on the parameter values in the configuration part. This sample program shows for CYT2B7 series.

Use case:

- Setting LVD1: threshold 4.7 V in falling detection
- Setting LVD2: threshold 4.9 V in rising detection
- Setting RAM mode: Retained mode
- Setting reset: Software reset

Figure 6 shows an example flow for the reset procedure in RAM0 with LVD interrupt.

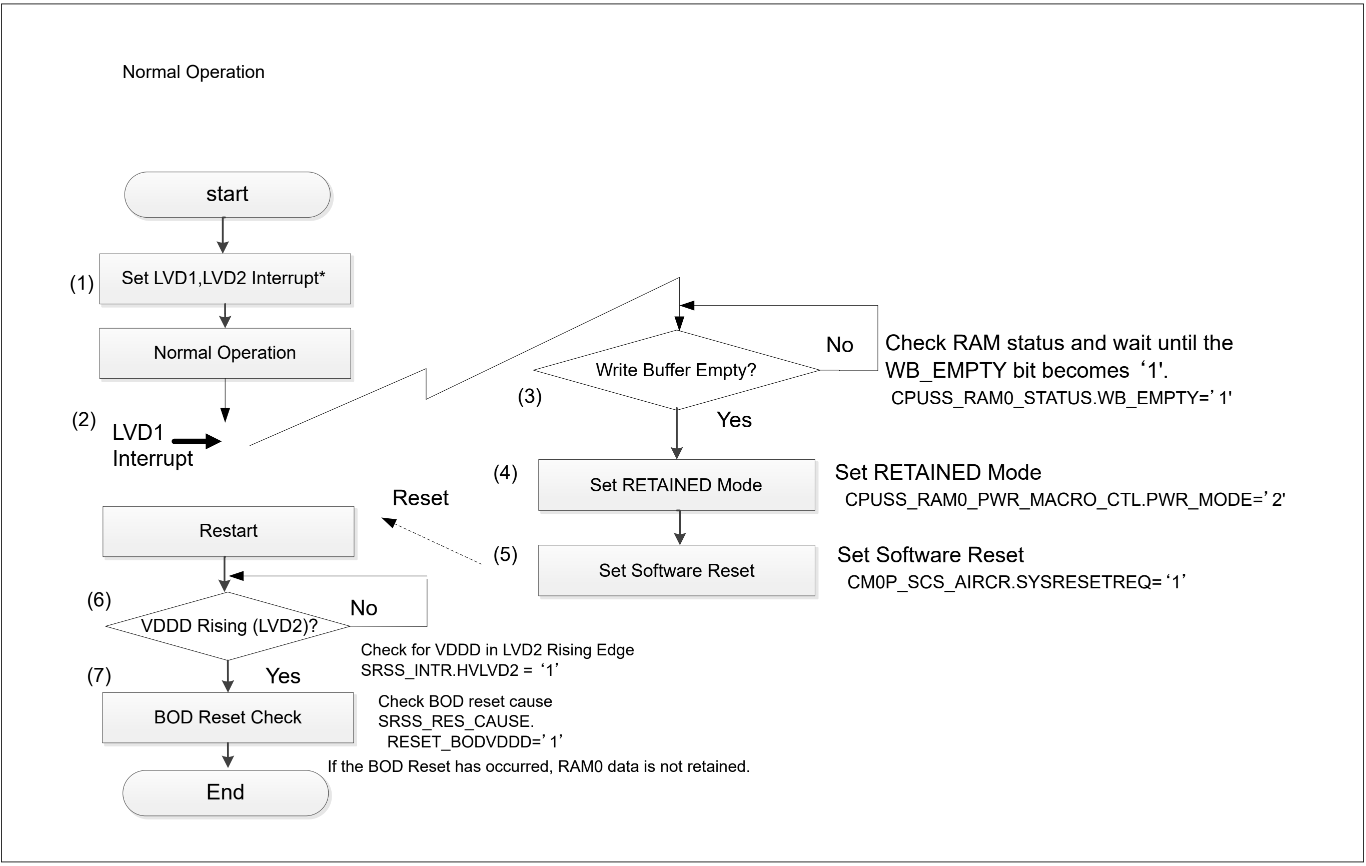

Figure 6 **Flowchart of example reset procedure in RAM0 with LVD interrupt**

*For details of the interrupt setting, see the "Interrupts" chapter of the architecture TRM.

1. Enable interrupt for LVD1 and LVD2.
2. If low voltage is detected with a voltage drop on VDDD, LVD1 generates an interrupt to the CPU.
3. When the CPU accepts the LVD interrupt, check the write buffer status.

   If there is data in the write buffer (WB_EMPTY = '0'), wait until the write buffer is empty.

4. When there is no data in the write buffer (WB_EMPTY = ‘1’), the CPU sets the RETAINED mode
5. The CPU issues a software reset.
6. The CPU checks to see whether the VDDD supply voltage has recovered.
7. If VDDD has recovered, the CPU verifies that a BOD reset has not occurred.If a BOD reset has not occurred, RAM0 data is retained. However, if a BOD reset has occurred, RAM0 data must be discarded.

Table 3 lists the parameters and Table 4 lists the functions of the configuration part in SDL for reset using LVD interrupt settings.

Table 3 List of initial reset using LVD interrupt parameters

Parameters	Description	Value
user_led0_port_pin_cfg. outVal	Pin output state	0ul
user_led0_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_STRONG_IN_OFF
user_led0_port_pin_cfg. Hsiom	Connection for I/O pin route	USER_LED0_PIN_MUX
user_led0_port_pin_cfg. intEdge	Edge which will trigger an IRQ	0ul
user_led0_port_pin_cfg. intMask	Masks edge interrupt	0ul
user_led0_port_pin_cfg. vtrip	Input buffer mode	0ul,
user_led0_port_pin_cfg. slewRate	Slew rate	0ul
user_led0_port_pin_cfg. driveSel	GPIO drive strength	0ul
user_led1_port_pin_cfg. outVal	Pin output state	0ul
user_led1_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_STRONG_IN_OFF
user_led1_port_pin_cfg. hsiom	Connection for I/O pin route	USER_LED1_PIN_MUX
user_led1_port_pin_cfg. intEdge	Edge which will trigger an IRQ	0ul
user_led1_port_pin_cfg. intMask	Masks edge interrupt	0ul
user_led1_port_pin_cfg. Vtrip	Input buffer mode	0ul
user_led1_port_pin_cfg. slewRate	Slew rate	0ul
user_led1_port_pin_cfg. driveSel	GPIO drive strength	0ul

Table 4 List of reset using LVD interrupt setting functions

Functions	Description	Value
Cy_SysReset_GetResetReason(void)	Returns the cause for the latest reset	-
Cy_LVD_ClearInterruptMask (cy_en_lvd_type_select_t lvdType)	Disables LVD interrupts. lvdType: LVD type	CY_LVD_TYPE_1 (for LVD1) CY_LVD_TYPE_2 (for LVD2)
Cy_LVD_SetThreshold (cy_en_lvd_type_select_t lvdType, cy_en_lvd_tripsel_t threshold)	Sets the threshold for monitoring the VDDD voltage. lvdType: LVD Type threshold: Threshold	CY_LVD_TYPE_1(for LVD1), CY_LVD_THRESHOLD_4_7_V (for LVD1) CY_LVD_TYPE_2 (for LVD2), CY_LVD_THRESHOLD_4_9_V (for LVD2)

(table continues...)

Table 4 (continued) List of reset using LVD interrupt setting functions

Functions	Description	Value
Cy_LVD_SetActionSelect (cy_en_lvd_type_select_t lvdType, cy_en_lvd_action_select_t lvdActionSelect)	Sets the action taken when the threshold is crossed in the programmed directions. lvdType: LVD Type lvdActionSelect: Action	CY_LVD_TYPE_1(for LVD1), CY_LVD_ACTION_INTR (for LVD1, LVD2) CY_LVD_TYPE_2(for LVD2),
Cy_LVD_SetEdgeSelect (cy_en_lvd_type_select_t lvdType, cy_en_lvd_edge_select_t lvdEdgeSelect)	Sets edge trigger when the threshold is crossed. lvdType: LVD Type lvdEdgeSelect: Edge selection	CY_LVD_TYPE_1(for LVD1), CY_LVD_EDGE_FALLING (for LVD1) CY_LVD_TYPE_2(for LVD2), CY_LVD_EDGE_RISING (for LVD2)
Cy_LVD_Enable (cy_en_lvd_type_select_t lvdType)	Enables the output of the LVD. lvdType: LVD Type	CY_LVD_TYPE_1(for LVD1) CY_LVD_TYPE_2(for LVD2)
Cy_LVD_SetInterrupt (cy_en_lvd_type_select_t lvdType)	Triggers the device to generate an interrupt for the LVD. lvdType: LVD Type	CY_LVD_TYPE_1(for LVD1) CY_LVD_TYPE_2(for LVD2)
void Cy_LVD_SetInterruptMask (cy_en_lvd_type_select_t lvdType)	Enables LVD interrupts. lvdType: LVD Type	CY_LVD_TYPE_1(for LVD1) CY_LVD_TYPE_2(for LVD2)
void Cy_LVD_ClearInterrupt (cy_en_lvd_type_select_t lvdType)	Clears LVD interrupts. lvdType: LVD Type	CY_LVD_TYPE_1(for LVD1) CY_LVD_TYPE_2(for LVD2)
Cy_Cpu_SramWriteBufferStatus (cy_en_cpu_sram_macro_t sramType)	Checks the write buffer status. sramType: SRAM type	CY_CPU_SRAM0
Cy_Cpu_SramPowerModeSet (cy_en_cpu_sram_macro_t sramType, cy_en_cpu_sram_power_mode_t pwrMode, uint8_t sramMacro)	Sets the SRAM power mode. sramType: SRAM type pwrMode: SRAM power mode sramMacro: Power control register index	CY_CPU_SRAM0, CY_CPU_SRAM_PM_RETAINED, 0ul

Code Listing 1 shows an example program of configuration part for the reset using LVD interrupt.

This application note does not list the functions related to the LED operation.

The following description will help you understand the register notation of the driver part of the SDL:

- SRSS-> unPWR_LVD_CTL register is the PWR_LVD_CTL register mentioned in the register TRM. Other registers are also described in the same manner.

See cyip_srss_v2.h under hdr/rev_x/ip for more information on the union and structure representation of registers.

Code Listing 1 Example program to reset using LVD interrupt

```
/* Define for LED0 */
/* Define the port settings */
#define USER_LED0_PORT            CY_LED0_PORT
#define USER_LED0_PIN             CY_LED0_PIN
#define USER_LED0_PIN_MUX         CY_LED0_PIN_MUX

/* Define for LED1 */
/* Define the port settings */
#define USER_LED1_PORT            CY_LED1_PORT
#define USER_LED1_PIN             CY_LED1_PIN
#define USER_LED1_PIN_MUX         CY_LED1_PIN_MUX


/* Set to gpio for LED0 */
/* Configure the port setting parameters for LED0. See Table 3. */
cy_stc_gpio_pin_config_t user_led0_port_pin_cfg =
{
  .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
  .hsiom     = USER_LED0_PIN_MUX,
  .intEdge   = 0ul,
  .intMask   = 0ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel  = 0ul,
};
/* Set to gpio for LED1 */
/* Configure the port setting parameters for LED1. See Table 3.*/
cy_stc_gpio_pin_config_t user_led1_port_pin_cfg =
{
  .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
  .hsiom     = USER_LED1_PIN_MUX,
  .intEdge   = 0ul,
  .intMask   = 0ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel  = 0ul,
};


/* Set to LVD interrupt */
/* Configure the interrupt structure parameters */
const  cy_stc_sysint_irq_t irq_cfg=
{
  .sysIntSrc  = srss_interrupt_IRQn,
  .intIdx     = CPUIntIdx2_IRQn,
  .isEnabled  = true,
};
/* VDDD checking flag */
bool VDDD_rising_check = false;
```

```
void LVD_IntHandler(void)    //Interrupt routine for LVD1 and LVD2
{
  uint32_t intStatus_4_7v;
  uint32_t intStatus_4_9v;
  uint32_t LVD_Status_4_7v_th;
  uint32_t LVD_Status_4_9v_th;

  /* LVD1 interrupt cause */
  intStatus_4_7v = Cy_LVD_GetInterruptStatusMasked(CY_LVD_TYPE_1);
  /* LVD2 interrupt cause */
  intStatus_4_9v = Cy_LVD_GetInterruptStatusMasked(CY_LVD_TYPE_2);

  /* If LVD_Status_4_7v_th is "1", it is above the threshold. If it is "0", it is below the
threshold. The LVD_Status_4_9v_th is also same. */
  LVD_Status_4_7v_th = Cy_LVD_GetStatus(CY_LVD_TYPE_1);
  LVD_Status_4_9v_th = Cy_LVD_GetStatus(CY_LVD_TYPE_2);

  /* Clear to interrupt LVD1 */
  Cy_LVD_ClearInterrupt(CY_LVD_TYPE_1);
  /* Clear to interrupt LVD2 */
  Cy_LVD_ClearInterrupt(CY_LVD_TYPE_2);

  /* LVD1 interrupt case */
  /* (2) LVD1 detection.*/
  if(((intStatus_4_7v >> 1) == 1ul) && (LVD_Status_4_7v_th == 0ul))
  {
    /* RAM status check */
    /* (3) Check the write buffer status. See Code Listing 11.*/
    while(0ul==(Cy_Cpu_SramWriteBufferStatus(CY_CPU_SRAM0)));

    /* RAM retain mode setting */
    /* (4) Sets the RETAINED mode. See Code Listing 12.*/
    Cy_Cpu_SramPowerModeSet(CY_CPU_SRAM0, CY_CPU_SRAM_PM_RETAINED, 0ul);

    /* Software reset */
    NVIC_SystemReset();     //(5) Set software reset.
  }



  /* LVD2 interrupt case */
  else if(((intStatus_4_9v >> 2) == 1ul) && (LVD_Status_4_9v_th ==  1ul) &&  ((intStatus_4_7v
>> 1) == 0ul))
  {
    /* VDDD was returned to rising point */
    VDDD_rising_check = true;
  }
  else
  {
  }
```

```
}


int main(void)
{
  uint32_t tRstReason = 0ul;

  __enable_irq();

  SystemInit();

  /* Read the RESET reason */
  /* (1) Set for LVD1, LVD2 interrupt.*/
  /*Read for reset cause in restart. See Code Listing 2.*/
  tRstReason = Cy_SysReset_GetResetReason();    //

  /* (1) Set for LVD1, LVD2 interrupt.*/
  /* Clear to interrupt mask for LVD1  See Code Listing 3*/
  Cy_LVD_ClearInterruptMask(CY_LVD_TYPE_1);


  /* LVD1,LVD2 settings */
  /* (1) Set for LVD1, LVD2 interrupt.*/
  /* Set the LVD1 threshold  See Code Listing 4.*/
  Cy_LVD_SetThreshold(CY_LVD_TYPE_1, CY_LVD_THRESHOLD_4_7_V);



  /* Action Select for LVD1 */
  /* (1) Set for LVD1, LVD2 interrupt.*/
  /* Set the action for LVD1. See Code Listing 5.*/
  Cy_LVD_SetActionSelect(CY_LVD_TYPE_1, CY_LVD_ACTION_INTR);



  /* (1) Set for LVD1, LVD2 interrupt.*/
  /* Edge Select for LVD1  See Code Listing 6.*/
  Cy_LVD_SetEdgeSelect(CY_LVD_TYPE_1, CY_LVD_EDGE_FALLING);



  /* (1) Set for LVD1, LVD2 interrupt.*/
  /* Enable LVD1  See Code Listing 7*/
  Cy_LVD_Enable(CY_LVD_TYPE_1);



  /* Set interupt for LVD1 */
  /* (1) Set for LVD1, LVD2 interrupt.*/
  Cy_LVD_SetInterrupt(CY_LVD_TYPE_1);     //Enable for LVD1 interrupt. See Code Listing 8.
  Cy_LVD_SetInterruptMask(CY_LVD_TYPE_1);    //Set to interrupt mask for LVD1. See Code Listing
9.
```

```
/* (1) Set for LVD1, LVD2 interrupt.*/
/* Clear to interrupt LVD1  See Code Listing 10*/
Cy_LVD_ClearInterrupt(CY_LVD_TYPE_1);



/* Clear to interrupt mask for LVD2 */
/* (1) Set for LVD1, LVD2 interrupt.*/
/* For LVD2 settings, refer to the LVD1.*/
Cy_LVD_ClearInterruptMask(CY_LVD_TYPE_2);

/* Set the LVD2 threshold */
/* (1) Set for LVD1, LVD2 interrupt.*/
/* For LVD2 settings, refer to the LVD1.*/
Cy_LVD_SetThreshold(CY_LVD_TYPE_2, CY_LVD_THRESHOLD_4_9_V);

/* Action Select for LVD2 */
/* (1) Set for LVD1, LVD2 interrupt.*/
/* For LVD2 settings, refer to the LVD1.*/
Cy_LVD_SetActionSelect(CY_LVD_TYPE_2, CY_LVD_ACTION_INTR);

/* Edge Select for LVD2 */
/* For LVD2 settings, refer to the LVD1.*/
Cy_LVD_SetEdgeSelect(CY_LVD_TYPE_2, CY_LVD_EDGE_RISING);

/* Enable LVD2 */
/* For LVD2 settings, refer to the LVD1.*/
Cy_LVD_Enable(CY_LVD_TYPE_2);

/* Set interupt for LVD1 */
/* For LVD2 settings, refer to the LVD1.*/
Cy_LVD_SetInterrupt(CY_LVD_TYPE_2);
Cy_LVD_SetInterruptMask(CY_LVD_TYPE_2);

/* Clear to interrupt LVD2 */
/* For LVD2 settings, refer to the LVD1.*/
Cy_LVD_ClearInterrupt(CY_LVD_TYPE_2);

/* Initialize in GPIO for LED */
/* Set for LED0, LED1.*/
Cy_GPIO_Pin_Init(USER_LED0_PORT, USER_LED0_PIN, &user_led0_port_pin_cfg);
Cy_GPIO_Pin_Init(USER_LED1_PORT, USER_LED1_PIN, &user_led1_port_pin_cfg);


/* Interrupt settings */
/* Set for interrupt routine.*/
Cy_SysInt_InitIRQ(&irq_cfg);
Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, LVD_IntHandler);
NVIC_SetPriority(CPUIntIdx2_IRQn, 0ul);
NVIC_ClearPendingIRQ(CPUIntIdx2_IRQn);
NVIC_EnableIRQ(CPUIntIdx2_IRQn);
```

```
  for(;;)
  {
    /* VDDD return checking point */
    /* (6) Check for VDDD risig in restart by LVD2 detection.*/
    if(VDDD_rising_check == true){

      /* After the LVD2 interrupt routine occurred,                 */
      /* when this program entered the code here, VDDD was returned.*/
      VDDD_rising_check = false;

      /* LED0 blink */
      for(uint16_t i = 0ul; i < 50ul; i++)
      {
        Cy_SysTick_DelayInUs(50000ul);
        Cy_GPIO_Inv(USER_LED0_PORT, USER_LED0_PIN);    //If VDDD was retuned in restart, the
LED0 blink.
      }
    }

    /* Check for BOD reset generation */
    if( ( tRstReason & CY_SYSRESET_BODVDDD ) == CY_SYSRESET_BODVDDD )    //(7) Check BOD reset
cause in restart.
    {
      /* If the BOD occurs, RAM data was not retained.*/
      /* LED1 turn on */
      Cy_GPIO_Write(USER_LED1_PORT, USER_LED1_PIN, 1);    //If BOD reset was generated, the
LED1 turn on.
    }
  }
}
```

Code Listing 2 to Code Listing 12 show an example program to configure reset using LVD interrupt in the driver part.

Code Listing 2 Example program to SysReset_GetResetReason in driver part

```
uint32_t Cy_SysReset_GetResetReason(void)
{
  return(SRSS->unRES_CAUSE.u32Register);    //(7) Check BOD reset cause.
}
```

Code Listing 3 Example program to clear interrupt mask in driver part

```
void Cy_LVD_ClearInterruptMask(cy_en_lvd_type_select_t lvdType)
{
  /* Clear the interrupt mask.*/
  if(lvdType == CY_LVD_TYPE_1)
  {
    SRSS->unSRSS_INTR_MASK.u32Register &= (uint32_t) ~SRSS_SRSS_INTR_MASK_HVLVD1_Msk;
  }
  else
  {
  SRSS->unSRSS_INTR_MASK.u32Register &= (uint32_t) ~SRSS_SRSS_INTR_MASK_HVLVD2_Msk;
  }
}
```

Code Listing 4 Example program to LVD SetThreshold in driver part

```
void Cy_LVD_SetThreshold(cy_en_lvd_type_select_t lvdType, cy_en_lvd_tripsel_t threshold)
{
  CY_ASSERT(CY_LVD_CHECK_TRIPSEL(threshold));

  if(lvdType == CY_LVD_TYPE_1)
  {
    /* Set the threshold.*/
    SRSS->unPWR_LVD_CTL.u32Register =  _CLR_SET_FLD32U(SRSS->unPWR_LVD_CTL.u32Register,
SRSS_PWR_LVD_CTL_HVLVD1_TRIPSEL_HT, threshold);
  }
  else
  {
    SRSS->unPWR_LVD_CTL2.u32Register = _CLR_SET_FLD32U(SRSS->unPWR_LVD_CTL2.u32Register,
SRSS_PWR_LVD_CTL2_HVLVD2_TRIPSEL_HT, threshold);
  }
}
```

Code Listing 5 Example program to LVD SetActionSelect in driver part

```
void Cy_LVD_SetActionSelect(cy_en_lvd_type_select_t lvdType, cy_en_lvd_action_select_t
lvdActionSelect)
{
  CY_ASSERT(CY_LVD_CHECK_ACTION_CFG(lvdActionSelect));

  if(lvdType == CY_LVD_TYPE_1)
  {
    /*Set the action.*/
    SRSS->unPWR_LVD_CTL.u32Register = _CLR_SET_FLD32U(SRSS->unPWR_LVD_CTL.u32Register,
SRSS_PWR_LVD_CTL_HVLVD1_ACTION, lvdActionSelect);
  }
  else
  {
    SRSS->unPWR_LVD_CTL2.u32Register = _CLR_SET_FLD32U(SRSS->unPWR_LVD_CTL2.u32Register,
SRSS_PWR_LVD_CTL2_HVLVD2_ACTION, lvdActionSelect);
  }
}
```

Code Listing 6 Example program to LVD SetEdgeSelect in driver part

```
void Cy_LVD_SetEdgeSelect(cy_en_lvd_type_select_t lvdType, cy_en_lvd_edge_select_t
lvdEdgeSelect)
{
  CY_ASSERT(CY_LVD_CHECK_EDGE_CFG(lvdEdgeSelect));

  if(lvdType == CY_LVD_TYPE_1)
  {
    /* Select the detection edge.*/
    SRSS->unPWR_LVD_CTL.u32Register = _CLR_SET_FLD32U(SRSS->unPWR_LVD_CTL.u32Register,
SRSS_PWR_LVD_CTL_HVLVD1_EDGE_SEL, lvdEdgeSelect);
  }
  else
  {
    SRSS->unPWR_LVD_CTL2.u32Register = _CLR_SET_FLD32U(SRSS->unPWR_LVD_CTL2.u32Register,
SRSS_PWR_LVD_CTL2_HVLVD2_EDGE_SEL, lvdEdgeSelect);
  }
}
```

Code Listing 7 Example program to LVD Enable in driver part

```
void Cy_LVD_Enable(cy_en_lvd_type_select_t lvdType)
{
  if(lvdType == CY_LVD_TYPE_1)
  {
    /* Enable for LVD1.*/
    SRSS->unPWR_LVD_CTL.u32Register |= SRSS_PWR_LVD_CTL_HVLVD1_EN_HT_Msk;
  }
  else
  {
    SRSS->unPWR_LVD_CTL2.u32Register |= SRSS_PWR_LVD_CTL2_HVLVD2_EN_HT_Msk;
  }
}
```

Code Listing 8 Example program to LVD SetInterrupt in driver part

```
void Cy_LVD_SetInterrupt(cy_en_lvd_type_select_t lvdType)
{
  if(lvdType == CY_LVD_TYPE_1)
  {
    /*Enable for LVD1 interrupt.*/
    SRSS->unSRSS_INTR_SET.u32Register = SRSS_SRSS_INTR_SET_HVLVD1_Msk;
  }
  else
  {
    SRSS->unSRSS_INTR_SET.u32Register = SRSS_SRSS_INTR_SET_HVLVD2_Msk;
  }
}
```

Code Listing 9 Example program to LVD SetInterruptMask in driver part

```
void Cy_LVD_SetInterruptMask(cy_en_lvd_type_select_t lvdType)
{
  if(lvdType == CY_LVD_TYPE_1)
  {
    /*Set the interrupt mask for LVD1.*/
    SRSS->unSRSS_INTR_MASK.u32Register |= SRSS_SRSS_INTR_MASK_HVLVD1_Msk;
  }
  else
  {
    SRSS->unSRSS_INTR_MASK.u32Register |= SRSS_SRSS_INTR_MASK_HVLVD2_Msk;
  }
}
```

Code Listing 10 Example program to LVD ClearInterrupt in driver part

```
void Cy_LVD_ClearInterrupt(cy_en_lvd_type_select_t lvdType)
{
if(lvdType == CY_LVD_TYPE_1)
  {
    /*Clear the interrupt.*/
    SRSS->unSRSS_INTR.u32Register = SRSS_SRSS_INTR_HVLVD1_Msk;
  }
  else
  {
    SRSS->unSRSS_INTR.u32Register = SRSS_SRSS_INTR_HVLVD2_Msk;
  }

  (void) SRSS->unSRSS_INTR.u32Register;
}
```

Code Listing 11 Example program to CPU SramWriteBufferStatus in driver part

```
bool Cy_Cpu_SramWriteBufferStatus(cy_en_cpu_sram_macro_t sramType)
{
  bool wb_status = false;

  switch(sramType)
  {
    case CY_CPU_SRAM0:
    /*(3) Check for buffer status*/
    wb_status = CPUSS->unRAM0_STATUS.stcField.u1WB_EMPTY;
    break;
    case CY_CPU_SRAM1:
    wb_status = CPUSS->unRAM1_STATUS.stcField.u1WB_EMPTY;
    break;
    case CY_CPU_SRAM2:
    wb_status = CPUSS->unRAM2_STATUS.stcField.u1WB_EMPTY;
    break;
    default:
    break;
  }

  return wb_status;
}
```

Code Listing 12 Example program to CPU SramPowerModeSet in driver part

```
void Cy_Cpu_SramPowerModeSet(cy_en_cpu_sram_macro_t sramType, cy_en_cpu_sram_power_mode_t
pwrMode, uint8_t sramMacro)
{
  switch(sramType)
  {
    case CY_CPU_SRAM0:
    /*(4) Sets the RETAINED mode.*/
    CPUSS->unRAM0_PWR_MACRO_CTL[sramMacro].stcField.u2PWR_MODE = pwrMode;
    break;
    case CY_CPU_SRAM1:
    CPUSS->unRAM1_PWR_CTL.stcField.u2PWR_MODE = pwrMode;
    break;
    case CY_CPU_SRAM2:
    CPUSS->unRAM2_PWR_CTL.stcField.u2PWR_MODE = pwrMode;
    break;
    default:
    break;
  }
}
```

3.2 Reset using external reset

In this case, low-voltage detection is performed by an external LVD IC. Figure 7 shows the block diagram for the reset procedure for RAM0 retention by using an input signal from an external LVD IC to a GPIO pin.

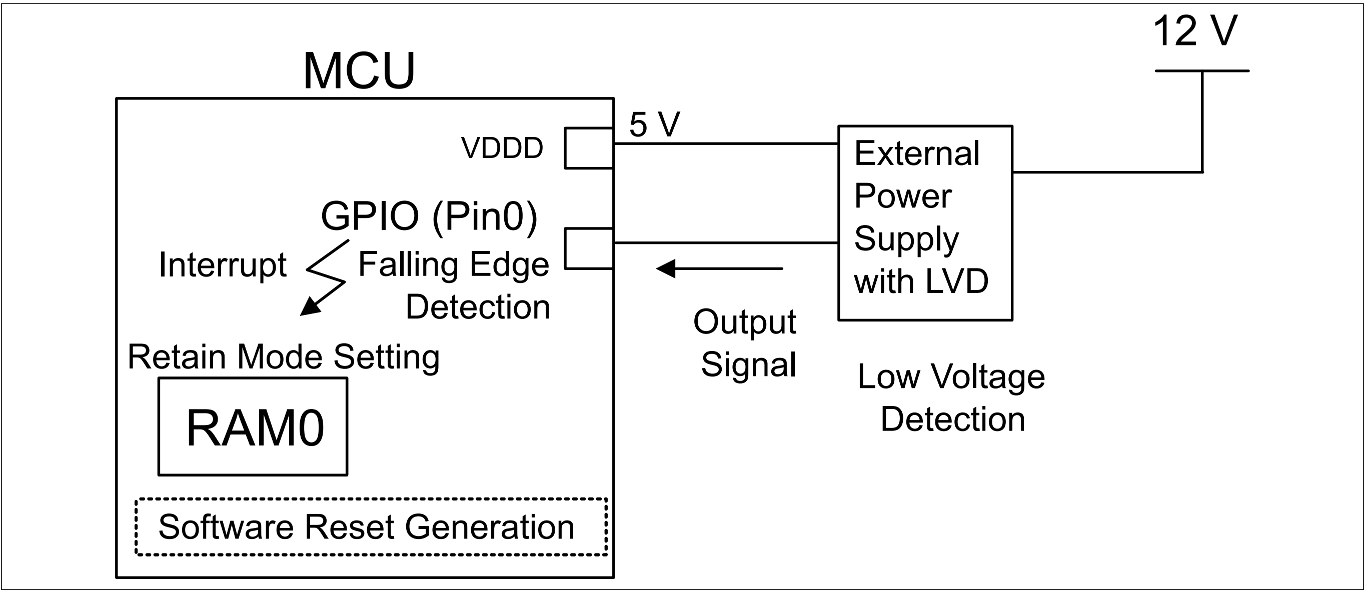

Figure 7 Block diagram for reset procedure in RAM0 retention by external LVD IC input signal to GPIO

In Figure 7, the output of this IC is connected to GPIO Pin0. Pin0 is configured as the interrupt pin, which can generate an interrupt, similar to the case of using the LVD. In the case of CYT3 and CYT4, where supply voltage is 5 V and 1.15 V, use an external LVD IC to monitor both 5 V and 1.15 V.

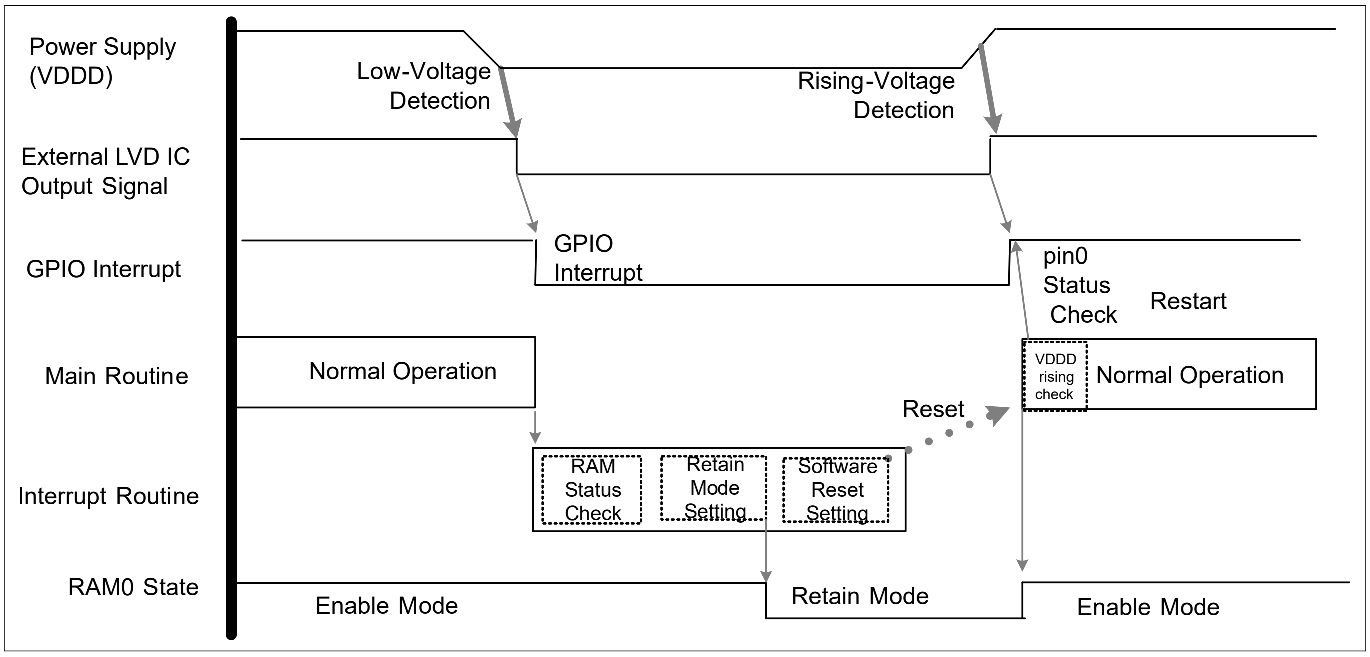

Figure 8 **Example of reset procedure timing chart in RAM0 with an external LVD IC**

In Figure 8, after detecting low voltage with an external LVD IC on VDDD, the external LVD IC output is made LOW. GPIO Pin0 detects this falling edge and generates an interrupt. When the interrupt occurs, the MCU operation transitions from the main routine to the interrupt routine. Then, the interrupt routine checks the RAM0 status, RETAINED mode setting, and software reset setting (reset generation). After the MCU starts up from a reset, check for VDDD rising with the GPIO Pin0 status because GPIO Pin0 is connected to the LVD IC output, and verify that a BOD reset has not occurred.

3.2.1 Use case

This section explains how to configure the Reset using external reset based on a use case using the Sample Driver Library (SDL). The code snippets in this application note are part of SDL. See Other references for the SDL.

Use case:

- Setting external input signal: Button pushing
- Setting external input port: P6_5
- Setting GPIO interrupt: falling input signal
- Setting RAM mode: Retained mode
- Setting reset: Software reset

Figure 9 shows an example flow for the reset Procedure in RAM0 retention with an external LVD IC input signal to GPIO.

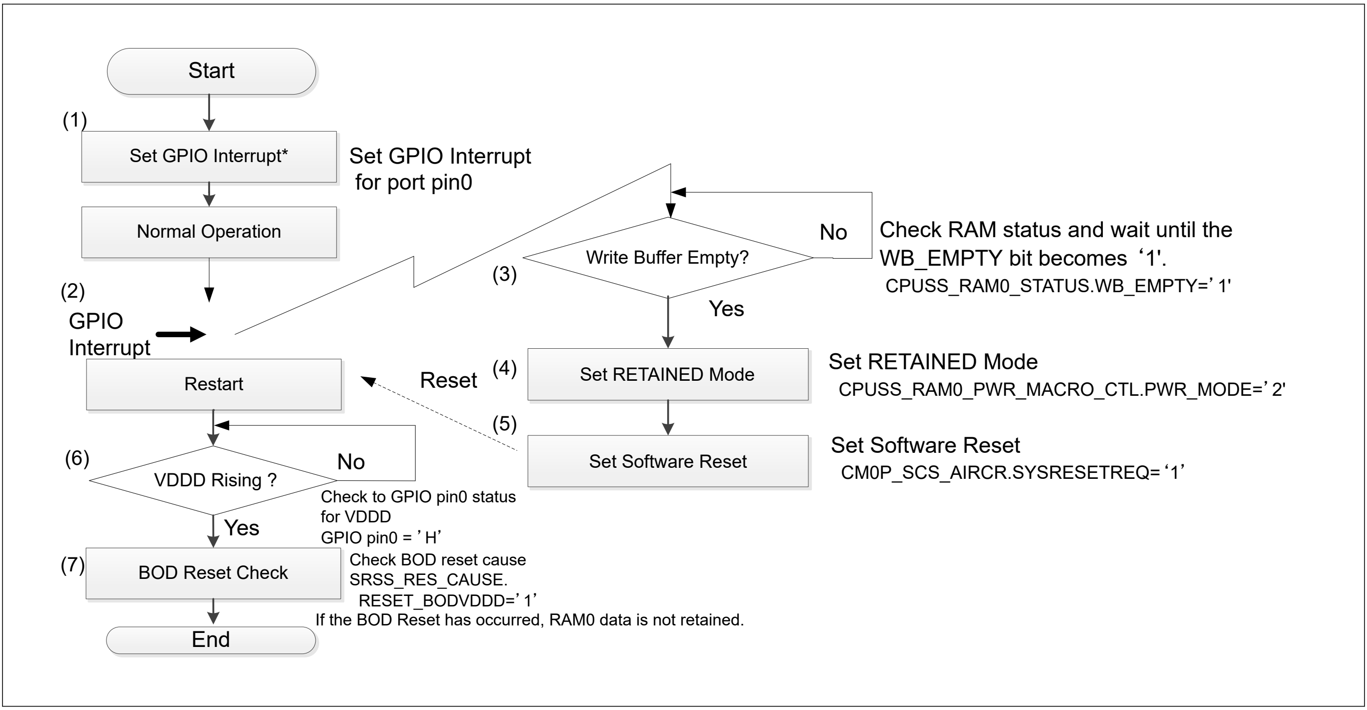

Figure 9 **Example of reset procedure in RAM0 retention with an external LVD IC input signal to GPIO**

*For details of interrupt setting, see the "Interrupts" chapter of the architecture TRM.

1. Enable interrupt for GPIO Pin0.
2. If GPIO Pin0 detects the input signal from the external LVD IC, GPIO generates an interrupt to the CPU.
3. When the CPU accepts the GPIO interrupt, check the write buffer status.
   If there is data in the write buffer (WB_EMPTY = '0'), wait until the write buffer is empty.
4. When there is no data in the write buffer (WB_EMPTY = '1'), the CPU sets the RETAINED mode.
5. The CPU issues a software reset.
6. The CPU verifies whether the VDDD supply voltage has recovered.
7. When VDDD has recovered, the CPU verifies that a BOD reset has not occurred.If a BOD reset has not occurred, the RAM0 data is retained. However, if a BOD reset has occurred, the RAM0 data must be discarded.

Table 5 lists the parameters and Table 6 lists the functions of the configuration part in SDL for reset using external reset settings.

Table 5 **List of initial reset using external reset parameters**

Parameters	Description	Value
user_button_port_pin_cfg. outVal	Pin output state	0ul
user_button_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_HIGHZ
user_button_port_pin_cfg. hsiom	Connection for I/O pin route	USER_BUTTON_PIN_MUX
user_button_port_pin_cfg. intEdge	Edge which will trigger an IRQ	CY_GPIO_INTR_FALLING
user_button_port_pin_cfg. intMask	Masks edge interrupt	1ul
user_button_port_pin_cfg. vtrip	Input buffer mode	0ul,
user_button_port_pin_cfg. slewRate	Slew rate	0ul

(table continues...)

Table 5 (continued) List of initial reset using external reset parameters

Parameters	Description	Value
user_button_port_pin_cfg. driveSel	GPIO drive strength	0ul
user_led0_port_pin_cfg. outVal	Pin output state	0ul
user_led0_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_STRONG_IN_OFF
user_led0_port_pin_cfg. hsiom	Connection for I/O pin route	USER_LED0_PIN_MUX
user_led0_port_pin_cfg. intEdge	Edge which will trigger an IRQ	0ul
user_led0_port_pin_cfg. intMask	Masks edge interrupt	0ul
user_led0_port_pin_cfg. vtrip	Input buffer mode	0ul,
user_led0_port_pin_cfg. slewRate	Slew rate	0ul
user_led0_port_pin_cfg. driveSel	GPIO drive strength	0ul
user_led1_port_pin_cfg. outVal	Pin output state	0ul
user_led1_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_STRONG_IN_OFF
user_led1_port_pin_cfg. hsiom	Connection for I/O pin route	USER_LED1_PIN_MUX
user_led1_port_pin_cfg. intEdge	Edge which will trigger an IRQ	0ul
user_led1_port_pin_cfg. intMask	Masks edge interrupt	0ul
user_led1_port_pin_cfg. Vtrip	Input buffer mode	0ul
user_led1_port_pin_cfg. slewRate	Slew rate	0ul
user_led1_port_pin_cfg. driveSel	GPIO drive strength	0ul

Table 6 List of reset using external reset setting functions

Functions	Description	Value
Cy_GPIO_Pin_Init (volatile stc_GPIO_PRT_t *base, uint32_t pinNum, const cy_stc_gpio_pin_config_t *config)	Initializes all pin configuration setting for the pin. *base: Pointer to the pin's port register base address pinNum: Position of the pin bit-field within the port register *config: Pointer to the pin config structure base address	USER_BUTTON_PORT, USER_BUTTON_PIN, &user_button_port_pin_cfg
Cy_SysReset_GetResetReason(void)	Returns the cause for the latest reset	-
Cy_Cpu_SramWriteBufferStatus (cy_en_cpu_sram_macro_t sramType)	Checks the write buffer status sramType: SRAM type	CY_CPU_SRAM0
Cy_Cpu_SramPowerModeSet (cy_en_cpu_sram_macro_t sramType, cy_en_cpu_sram_power_mode_t pwrMode, uint8_t sramMacro)	Sets the SRAM power mode sramType: SRAM type pwrMode: SRAM power mode sramMacro: Power control register index	CY_CPU_SRAM0, CY_CPU_SRAM_PM_RETAINED, 0ul

Code Listing 13 shows an example program of configuration part for the reset using external reset.

Code Listing 13 Example program to reset using external reset

```
/* Define GPIO button (P6_5) */
/* This button is used as an input signal assumed external LVD IC. */
/* Define the port settings.*/
#define USER_BUTTON_PORT        CY_CB_BUTTON_PORT
#define USER_BUTTON_PIN         CY_CB_BUTTON_PIN
#define USER_BUTTON_PIN_MUX     CY_CB_BUTTON_PIN_MUX
#define USER_BUTTON_IRQ         CY_CB_BUTTON_IRQN

/* Define for LED0 */
/* Define the port settings.*/
#define USER_LED0_PORT          CY_LED0_PORT
#define USER_LED0_PIN           CY_LED0_PIN
#define USER_LED0_PIN_MUX       CY_LED0_PIN_MUX

/* Define for LED1 */
/* Define the port settings.*/
#define USER_LED1_PORT          CY_LED1_PORT
#define USER_LED1_PIN           CY_LED1_PIN
#define USER_LED1_PIN_MUX       CY_LED1_PIN_MUX

/* Set to button for input signal  */
/* Configure the port setting parameters for button. See Table 5.*/
const cy_stc_gpio_pin_config_t user_button_port_pin_cfg =
{
  .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_HIGHZ,
  .hsiom     = USER_BUTTON_PIN_MUX,
  .intEdge   = CY_GPIO_INTR_FALLING,
  .intMask   = 1ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel  = 0ul,
};




/* LED0 gpio configuration */
/* Configure the port setting parameters for LED0. See Table 5.*/
cy_stc_gpio_pin_config_t user_led0_port_pin_cfg =
{
  .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
  .hsiom     = CY_LED0_PIN_MUX,
  .intEdge   = 0ul,
  .intMask   = 0ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel  = 0ul,
};
```

```
/* LED1 gpio configuration */
/*Configure the port setting parameters for LED1. See Table 5.*/
cy_stc_gpio_pin_config_t user_led1_port_pin_cfg =
{
  .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
  .hsiom     = CY_LED1_PIN_MUX,
  .intEdge   = 0ul,
  .intMask   = 0ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel  = 0ul,
};

/* Set to GPIO Button interrupt for falling edge */
/* Configure the interrupt structure parameters.*/
const cy_stc_sysint_irq_t irq_cfg =
{
  .sysIntSrc  = USER_BUTTON_IRQ,
  .intIdx     = CPUIntIdx3_IRQn,
  .isEnabled  = true,
};

void GPIO_IntHandler(void)    //(2) GPIO Interrupt routine.
{
  uint32_t intStatus;

  /* If falling edge detected */
  intStatus = Cy_GPIO_GetInterruptStatusMasked(USER_BUTTON_PORT, USER_BUTTON_PIN);
  if (intStatus != 0ul)
  {
    /* RAM staus check */
    /*(3) Check the write buffer status. See Code Listing 11.*/
    while(0ul==(Cy_Cpu_SramWriteBufferStatus(CY_CPU_SRAM0)));



    /* RAM retain mode setting */
    /*(4) Set the RETAINED mode. See Code Listing 12.*/
    Cy_Cpu_SramPowerModeSet(CY_CPU_SRAM0, CY_CPU_SRAM_PM_RETAINED, 0ul);

    /* Software reset */
    /* (5) Set software reset.*/
    NVIC_SystemReset();

    /* Clear to interrupt */
    Cy_GPIO_ClearInterrupt(USER_BUTTON_PORT, USER_BUTTON_PIN);
  }
}



int main(void)
```

```
{
  uint32_t tRstReason = 0ul;

  __enable_irq(); /* Enable global interrupts. */

  SystemInit();

  /* Port initialization */
  /*Set to GPIO for button settings in assuming signal input from an external LVD IC. See Code
Listing 14.*/
  Cy_GPIO_Pin_Init(USER_BUTTON_PORT, USER_BUTTON_PIN, &user_button_port_pin_cfg);

  /* LED port initialization */
  /* Set for LED0, LED1.*/
  Cy_GPIO_Pin_Init(USER_LED0_PORT, USER_LED0_PIN, &user_led0_port_pin_cfg);
  Cy_GPIO_Pin_Init(USER_LED1_PORT, USER_LED1_PIN, &user_led1_port_pin_cfg);


  /* Read the RESET reason */
  /* Check to reset cause in restart. See Code Listing 2.*/
  tRstReason = Cy_SysReset_GetResetReason();



  /* Check for software reset generation by LVD */
  if( ( tRstReason & CY_SYSRESET_SOFT ) == CY_SYSRESET_SOFT )
  {
    /* VDDD rising check */
    /* (6) Check to GPIO button pin for VDDD rising in restart.*/
    while(0ul==(Cy_GPIO_Read(USER_BUTTON_PORT, USER_BUTTON_PIN)));



    /* LED1 turn on */
    /* In the VDDD return, LED1 turn on .*/
    Cy_GPIO_Write(USER_LED1_PORT, USER_LED1_PIN, 1ul);
  }

  /* Check for BOD reset generation */
  /* (7) Check BOD reset cause in restart.*/
  if( ( tRstReason & CY_SYSRESET_BODVDDD ) == CY_SYSRESET_BODVDDD )
  {
    /* When the here code entered, RAM data was not retained.*/
  }

  /* GPIO interrupt settings */
  /* (1) Set GPIO interrupt.*/
  Cy_SysInt_InitIRQ(&irq_cfg);
  Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, GPIO_IntHandler);
  NVIC_SetPriority(irq_cfg.intIdx, 3ul);
  NVIC_EnableIRQ(irq_cfg.intIdx);

  for(;;)
```

```
    {
      /* Waiting 0.5 [s] in the LED0 */
      /* The LED0 blink in normal operation.*/
      Cy_SysTick_DelayInUs(500000ul);
      Cy_GPIO_Inv(USER_LED0_PORT, USER_LED0_PIN);
    }
  }
```

Code Listing 14 shows an example program to configure reset using external reset in the driver part.

Code Listing 14 Example program to GPIO pin init in the driver part

```
cy_en_gpio_status_t Cy_GPIO_Pin_Init(volatile stc_GPIO_PRT_t *base, uint32_t pinNum, const
cy_stc_gpio_pin_config_t *config)
{
/* Set for GPIO pin.*/
  cy_en_gpio_status_t status = CY_GPIO_SUCCESS;

  if((NULL != base) && (NULL != config))
  {
    Cy_GPIO_Write(base, pinNum, config->outVal);
    Cy_GPIO_SetHSIOM(base, pinNum, config->hsiom);
    Cy_GPIO_SetVtrip(base, pinNum, config->vtrip);
    Cy_GPIO_SetSlewRate(base, pinNum, config->slewRate);
    Cy_GPIO_SetDriveSel(base, pinNum, config->driveSel);
    Cy_GPIO_SetDrivemode(base, pinNum, config->driveMode);
    Cy_GPIO_SetInterruptEdge(base, pinNum, config->intEdge);
    Cy_GPIO_ClearInterrupt(base, pinNum);
    Cy_GPIO_SetInterruptMask(base, pinNum, config->intMask);
  }
  else
  {
    status = CY_GPIO_BAD_PARAM;
  }

  return(status);
}
```

4 RAM retention procedure in low-power mode

This section shows an example procedure for low-power mode transition with a block diagram, timing chart, and flowchart.

The MCU has a device power mode. Device power mode has Active mode, Sleep mode, DeepSleep mode, and Hibernate mode. Sleep mode, DeepSleep mode, and Hibernate mode are low-power modes; this application note describes the DeepSleep mode case.

See the "Device Power Modes" chapter of the Architecture TRM.

4.1 Using DeepSleep mode

In this case, DeepSleep mode transition is used. The setting of DeepSleep mode is performed in the Active mode in which the main program is running.

Figure 10 shows the block diagram for the DeepSleep mode transition procedure for RAM0 retention.

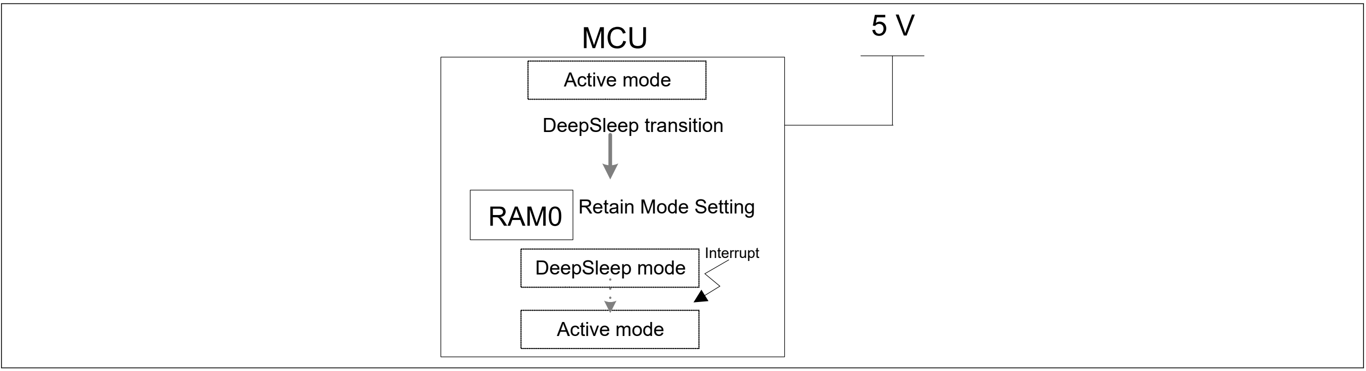

Figure 10 **Block diagram for DeepSleep mode transition procedure for RAM0 retention**

In Figure 10, first, transition to DeepSleep mode from Active mode. During this transition, check the RAM0 status and set the RETAINED mode. Then, the MCU enters DeepSleep mode. If an interrupt occurs while in DeepSleep mode, the MCU returns to Active mode.

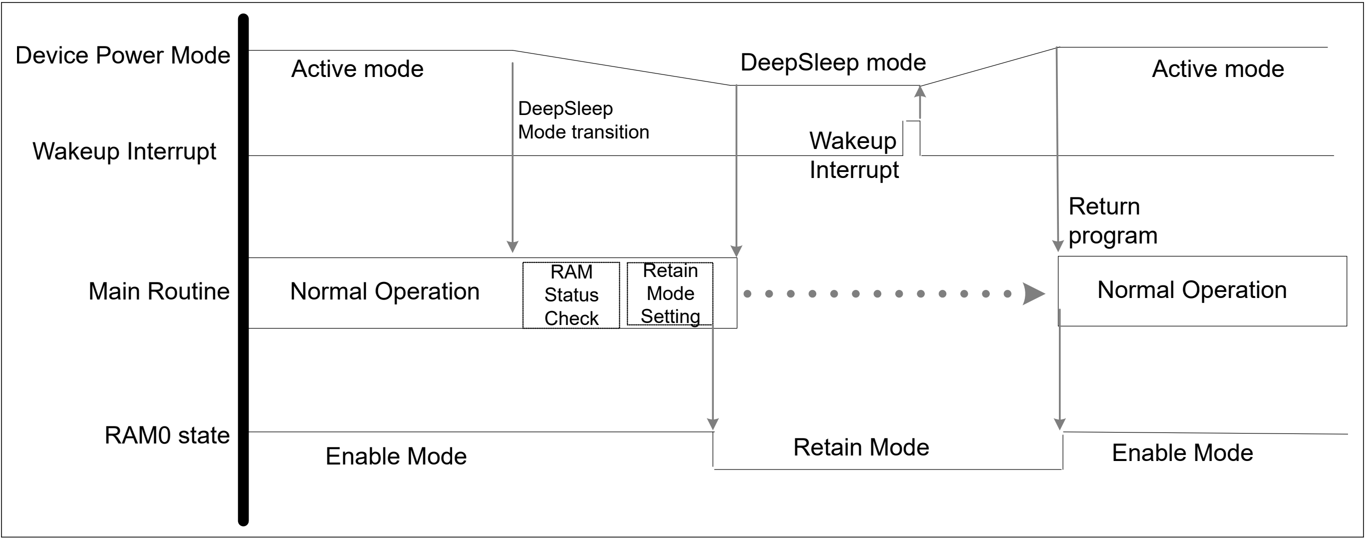

Figure 11 **Example of timing chart of DeepSleep mode transition procedure for RAM0 retention**

In Figure 11, when transitioning from Active mode to DeepSleep mode, the main routine checks the RAM0 status and sets the RETAINED mode. After that, the MCU goes into DeepSleep mode and the main routine stops. Then, in DeepSleep mode, when an interrupt occurs, it returns to Active mode. The main routine resumes program execution.

4.1.1 Use case

This section explains how to configure the DeepSleep mode transition based on a use case using the Sample Driver Library (SDL). The code snippets in this application note are part of SDL. See Other references for the SDL.

Use case:

- Setting transition low power mode: DeepSleep mode
- Setting RAM mode: Retained mode
- Setting return to Active mode: Wakeup input signal interrupt
- Setting wakeup input port: P6_5

Figure 12 shows an example flow for DeepSleep mode transition procedure for RAM0 retention.

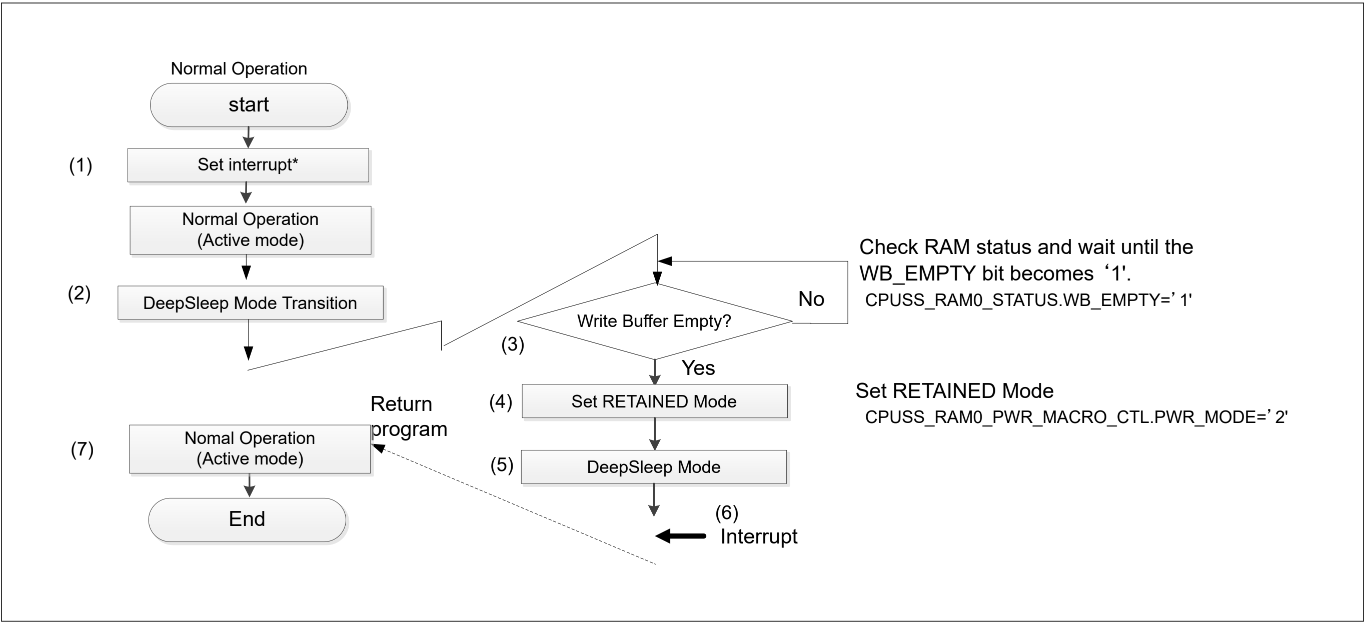
Normal Operation
start
(1)
Set interrupt*
Normal Operation (Active mode)
(2)
DeepSleep Mode Transition
Check RAM status and wait until the WB_EMPTY bit becomes '1'.
CPUSS_RAM0_STATUS.WB_EMPTY=' 1'
No
Write Buffer Empty?
(3)
Yes
Set RETAINED Mode
CPUSS_RAM0_PWR_MACRO_CTL.PWR_MODE=' 2'
(4)
Set RETAINED Mode
Return program
(7)
Nomal Operation (Active mode)
(5)
DeepSleep Mode
End
(6)
Interrupt

Figure 12 Flowchart of example DeepSleep mode transition procedure for RAM0 retention

*For details of interrupt setting, see the "Interrupts" chapter of the Architecture TRM.

1. Set interrupt.
2. When in Active mode of Normal operation, set Sleep mode.
3. Check the write buffer status. If there is data in the write buffer (WB_EMPTY = '0'), wait until the write buffer is empty.
4. When there is no data in the write buffer (WB_EMPTY = '1'), the CPU sets the RETAINED mode.
5. MCU enters DeepSleep mode.
6. When an interrupt occurs, MCU changes from DeepSleep mode to Active mode.
7. Program execution resumes.

Table 7 lists the parameters and Table 8 lists the functions of the configuration part in SDL for DeepSleep mode transition settings.

Table 7 List of initial DeepSleep mode transition parameters

Parameters	Description	Value
user_button_port_pin_cfg. outVal	Pin output state	0ul
user_button_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_HIGHZ

(table continues...)

Table 7 (continued) List of initial DeepSleep mode transition parameters

Parameters	Description	Value
user_button_port_pin_cfg. hsiom	Connection for I/O pin route	USER_BUTTON_PIN_MUX
user_button_port_pin_cfg. intEdge	Edge which will trigger an IRQ	CY_GPIO_INTR_FALLING
user_button_port_pin_cfg. intMask	Masks edge interrupt	1ul
user_button_port_pin_cfg. vtrip	Input buffer mode	0ul,
user_button_port_pin_cfg. slewRate	Slew rate	0ul
user_button_port_pin_cfg. driveSel	GPIO drive strength	0ul
user_led_port_pin_cfg. outVal	Pin output state	0ul
user_led_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_STRONG_IN_OFF
user_led_port_pin_cfg. hsiom	Connection for I/O pin route	USER_LED_PIN_MUX
user_led_port_pin_cfg. intEdge	Edge which will trigger an IRQ	0ul
user_led_port_pin_cfg. intMask	Masks edge interrupt	0ul
user_led_port_pin_cfg. vtrip	Input buffer mode	0ul
user_led_port_pin_cfg. slewRate	Slew rate	0ul
user_led_port_pin_cfg. driveSel	GPIO drive strength	0ul

Table 8 List of initial DeepSleep mode transition setting function

Functions	Description	Value
Cy_GPIO_Pin_Init (volatile stc_GPIO_PRT_t *base, uint32_t pinNum, const cy_stc_gpio_pin_config_t *config)	Initializes all pin configuration setting for the pin. *base: Pointer to the pin's port register base address pinNum: Position of the pin bit-field within the port register *config: Pointer to the pin config structure base address	USER_BUTTON_PORT, USER_BUTTON_PIN, &user_button_port_pin_cfg
Cy_Cpu_SramWriteBufferStatus (cy_en_cpu_sram_macro_t sramType)	sramType: SRAM type	CY_CPU_SRAM0
Cy_Cpu_SramPowerModeSet (cy_en_cpu_sram_macro_t sramType, cy_en_cpu_sram_power_mode_t pwrMode, uint8_t sramMacro)	Sets the SRAM power mode sramType: SRAM type pwrMode: SRAM power mode sramMacro: Power control register index	CY_CPU_SRAM0, CY_CPU_SRAM_PM_RETAINED, 0ul
Cy_SysPm_DeepSleep (cy_en_syspm_waitfor_t waitFor)	Sets a CPU core to the Deep Sleep mode. waitFor: Selects wait for action.	CY_SYSPM_WAIT_FOR_INTERRUPT

Code Listing 15 shows an example program of configuration part for the DeepSleep mode transiton.

Code Listing 15 Example program to DeepSleep mode transition

```
/* Define for Deepsleep wakeup gpio (P6_5) */
/* Define the port settings*/
#define USER_BUTTON_PORT        CY_CB_BUTTON_PORT
#define USER_BUTTON_PIN         CY_CB_BUTTON_PIN
#define USER_BUTTON_PIN_MUX     CY_CB_BUTTON_PIN_MUX
#define USER_BUTTON_IRQ         CY_CB_BUTTON_IRQN

/* Define for LED0 */
/* Define the port settings*/
#define USER_LED_PORT           CY_LED0_PORT
#define USER_LED_PIN            CY_LED0_PIN
#define USER_LED_PIN_MUX        CY_LED0_PIN_MUX

/* Set to wakeup pin for Deepsleep mode return  */
const cy_stc_gpio_pin_config_t user_button_port_pin_cfg =
{
/* Configure the port setting parameters for wakeup. See Table 7.*/
  .outVal     = 0ul,
  .driveMode = CY_GPIO_DM_HIGHZ,
  .hsiom      = USER_BUTTON_PIN_MUX,
  .intEdge    = CY_GPIO_INTR_FALLING,
  .intMask    = 1ul,
  .vtrip      = 0ul,
  .slewRate   = 0ul,
  .driveSel   = 0ul,
};

/* Set to gpio for LED0 */
/* Configure the port setting parameters for LED0. See Table 7.*/
cy_stc_gpio_pin_config_t user_led_port_pin_cfg =
{
  .outVal     = 0ul,
  .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
  .hsiom      = USER_LED_PIN_MUX,
  .intEdge    = 0ul,
  .intMask    = 0ul,
  .vtrip      = 0ul,
  .slewRate   = 0ul,
  .driveSel   = 0ul,
};

/* Set to GPIO Button interrupt for Deepsleep wakeup */
/* Configure the interrupt structure parameters.*/
const cy_stc_sysint_irq_t irq_cfg =
{
  .sysIntSrc  = USER_BUTTON_IRQ,
  .intIdx     = CPUIntIdx3_IRQn,
  .isEnabled  = true,
};

/* Deepsleep mode start flag */
```

```
uint8_t Deepsleep_transition = false;

/* Deepsleep mode transition checking */
cy_en_syspm_status_t return_value;


void ButtonIntHandler(void)    //(6) Interrupt routine
{
  uint32_t intStatus;

  /* If falling edge detected */
  intStatus = Cy_GPIO_GetInterruptStatusMasked(USER_BUTTON_PORT, USER_BUTTON_PIN);
  if (intStatus != 0ul)
  {
    /* Clear to interrupt flag */
    Cy_GPIO_ClearInterrupt(USER_BUTTON_PORT, USER_BUTTON_PIN);
  }
}


int main(void)
{
  __enable_irq(); /* Enable global interrupts. */

  SystemInit();

  /* Initialize in GPIO for Deeplsleep wakeup */
  /* Set GPIO pin for DeepSleep wakeup. See Code Listing 14.*/
  Cy_GPIO_Pin_Init(USER_BUTTON_PORT, USER_BUTTON_PIN, &user_button_port_pin_cfg);

  /* Initialize in GPIO for LED0 */
  /* Set for LED0.*/
  Cy_GPIO_Pin_Init(USER_LED_PORT, USER_LED_PIN, &user_led_port_pin_cfg);

  /* Interrupt settings */
  /* (1) Set interrupt.*/
  Cy_SysInt_InitIRQ(&irq_cfg);
  Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, ButtonIntHandler);
  NVIC_SetPriority(irq_cfg.intIdx, 3ul);
  NVIC_EnableIRQ(irq_cfg.intIdx);

  /* Deepsleep mode transition start setting */
  /* (2) Set for Deepsleep mode transition.*/
  Deepsleep_transition = true;



  /* SRAM write buffer status check */
  /* (3) Check the write buffer status. See Code Listing 11.*/
  while(0ul==(Cy_Cpu_SramWriteBufferStatus(CY_CPU_SRAM0)));

  /* RAM retain mode setting */
  /* (4) Sets the RETAINED mode. See Code Listing 12.*/
```

```
    Cy_Cpu_SramPowerModeSet(CY_CPU_SRAM0, CY_CPU_SRAM_PM_RETAINED, 0ul);

    for(;;)
    {
      /* Set to transfer to Deepsleep mode only once */
      if( Deepsleep_transition == true)
      {
        Deepsleep_transition = false;

        /* Transfer to Deepsleep mode */
        /* (5) MCU enters DeepSleep mode. See Code Listing 16.*/
        Cy_SysPm_DeepSleep((cy_en_syspm_waitfor_t)CY_SYSPM_WAIT_FOR_INTERRUPT);
      }

      /* Continuously blink an LED0 to indicate waking up from DeepSleep */
      /* (7) Program execution resumes. The LED0 blink.*/
      Cy_SysTick_DelayInUs(50000ul);
      Cy_GPIO_Inv(USER_LED_PORT, USER_LED_PIN);
    }
  }
```

Code Listing 16 show an example program to configure DeepSleep mode transition in the driver part.

Code Listing 16 Example program to SysPm_DeepSleep in driver part

```
void cy_en_syspm_status_t Cy_SysPm_DeepSleep(cy_en_syspm_waitfor_t waitFor)
{
  uint32_t interruptState;
  cy_en_syspm_status_t retVal = CY_SYSPM_SUCCESS;

  /* Call the registered callback functions with
  * the CY_SYSPM_CHECK_READY parameter.
  */
  if(0u != currentRegisteredCallbacksNumber)
  {
    retVal = Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_READY);
  }

  /* The device (core) can switch into the deep sleep power mode only when
  *  all executed registered callback functions with the CY_SYSPM_CHECK_READY
  *  parameter returned CY_SYSPM_SUCCESS.
  */
  if(retVal == CY_SYSPM_SUCCESS)
  {
    /* Call the registered callback functions with the CY_SYSPM_BEFORE_TRANSITION
    * parameter. The return value is ignored.
    */
    interruptState = Cy_SysLib_EnterCriticalSection();
    if(0u != currentRegisteredCallbacksNumber)
    {
      (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_BEFORE_ENTER);
    }

    #if(0u != CY_CPU_CORTEX_M0P)


    /* The CPU enters the deep sleep mode upon execution of WFI/WFE */
    SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;    //(5) MCU enters DeepSleep mode.

    if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
    {
      __WFI();
    }
    else
    {
      __WFE();
    }
    #else

    /* Repeat WFI/WFE instructions if wake up was not intended.
    */
    do
    {
      SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;

      if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
```

```
        {
          __WFI();
        }
        else
        {
          __WFE();
        }
      } while (0);

      #endif /* (0u != CY_CPU_CORTEX_M0P) */

      Cy_SysLib_ExitCriticalSection(interruptState);

      /* Call the registered callback functions with the CY_SYSPM_AFTER_TRANSITION
      *  parameter. The return value is ignored.
      */
      if(0u != currentRegisteredCallbacksNumber)
      {
        (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_AFTER_EXIT);
      }
    }
    else
    {
      /* Execute callback functions with the CY_SYSPM_CHECK_FAIL parameter to
      * undo everything done in the callback with the CY_SYSPM_CHECK_READY
      * parameter. The return value is ignored.
      */
      (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_FAIL);
      retVal = CY_SYSPM_FAIL;
    }
    return retVal;
  }
```

4.2 Using Hibernate mode

In this case, Hibernate mode transition is used. The setting of Hibernate mode is performed in the Active mode in which the main program is running.

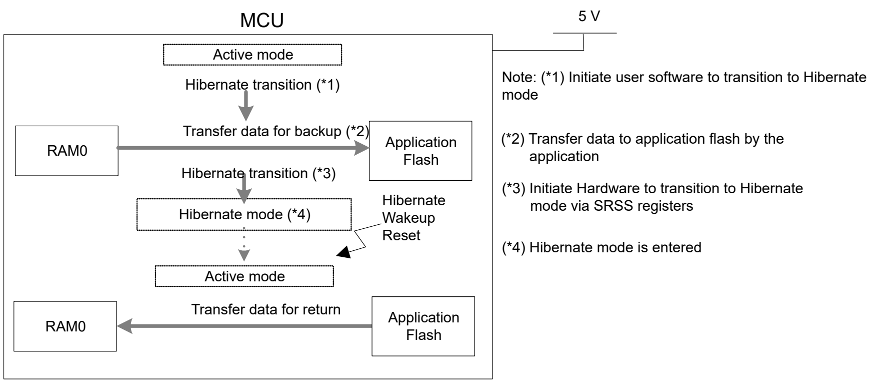
MCU
5 V
Active mode
Hibernate transition (*1)
Transfer data for backup (*2)
RAM0
Application Flash
Hibernate transition (*3)
Hibernate mode (*4)
Hibernate Wakeup Reset
Active mode
Transfer data for return
RAM0
Application Flash
Note: (*1) Initiate user software to transition to Hibernate mode
(*2) Transfer data to application flash by the application
(*3) Initiate Hardware to transition to Hibernate mode via SRSS registers
(*4) Hibernate mode is entered

Figure 13 **Block diagram for backup procedure in Hibernate mode transition for backup memory data**

Figure 13 shows the transition to Hibernate mode from Active mode. During this transition, access to the application flash is checked. If the application flash can be accessed, the backup memory data from the RAM is transferred to the application flash. After the data transition is completed, the user software configures the SRSS register to make the MCU enter Hibernate mode. If a Hibernate wakeup reset occurs while in Hibernate mode, the MCU returns to Active mode. After moving to Active mode, the backup memory data is transferred from the application flash to RAM.

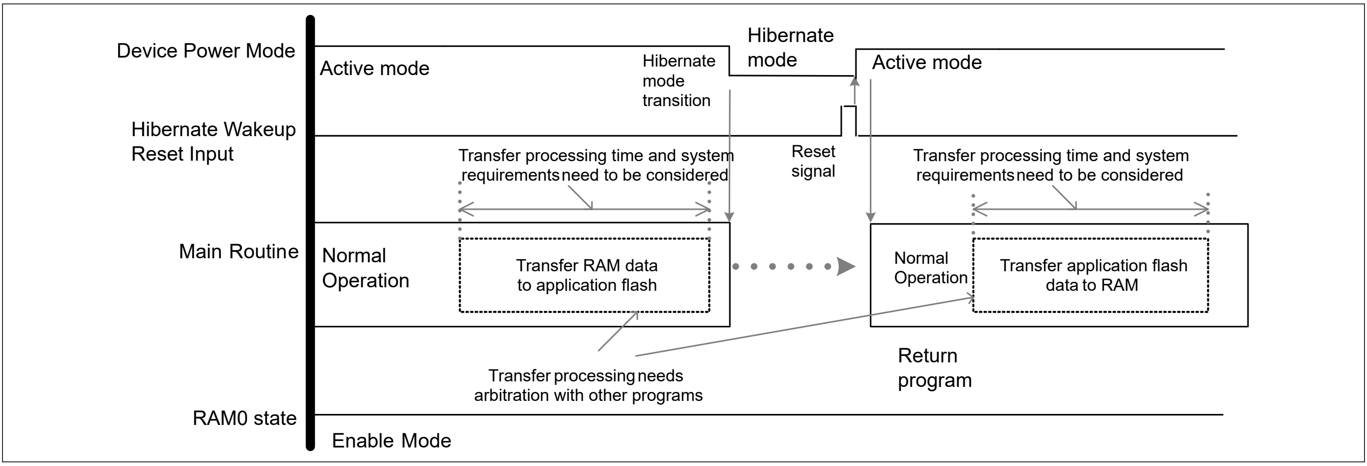
Device Power Mode
Active mode
Hibernate mode transition
Hibernate mode
Active mode
Hibernate Wakeup Reset Input
Reset signal
Transfer processing time and system requirements need to be considered
Transfer processing time and system requirements need to be considered
Main Routine
Normal Operation
Transfer RAM data to application flash
Normal Operation
Transfer application flash data to RAM
Transfer processing needs arbitration with other programs
Return program
RAM0 state
Enable Mode

Figure 14 **Example of timing chart of backup procedure in Hibernate mode transition for backup memory data**

In Figure 14, when transitioning from Active mode to Hibernate mode, the main program transfers the backup memory data from the RAM to application flash. While writing the application flash, the user must arbitrate to access. This arbitration is between the RAM and application flash backup access and other application program access. This arbitration is to prevent access of other programs during backup access.

After the arbitration, the MCU goes into Hibernate mode and the execution of the main routine stops. In Hibernate mode, when a Hibernate wakeup reset occurs, the MCU returns to Active mode. The main routine resumes program execution; the backup memory data is transferred from the application flash to RAM. In this transition, the arbitration with other application programs must be taken into account.

At the time of mode transition, the RAM and application flash transition time for data backup and return must meet the system requirements.

4.2.1 Use case

This section explains how to configure the Hibernate mode transition for backup memory data based on a use case using the Sample Driver Library (SDL). The code snippets in this application note are part of SDL. See Other references for the SDL.

Use case:

- Setting transition low power mode: Hibernate mode
- Setting return to Active mode: Hibernate wakeup reset
- Setting Hibernate wakeup reset port: P21_4
- Setting between RAM and application flash transition data: 0x5A5A5A5A
- Setting RAM address: 0x08000818
- Setting application flash address: 0x14000010

Figure 15 shows an example flow for Hibernate mode transition for backup memory data.

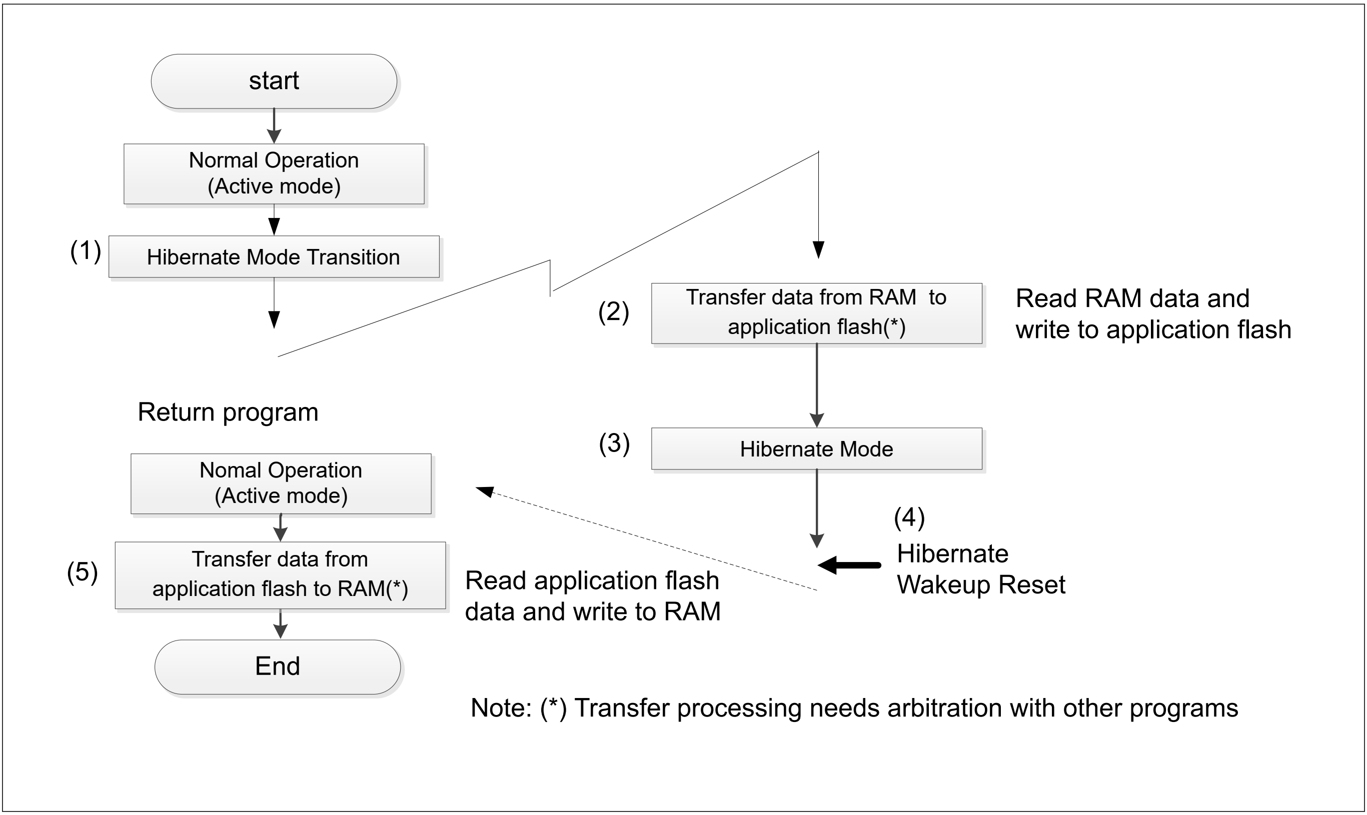
start
Normal Operation (Active mode)
(1)
Hibernate Mode Transition
(2)
Transfer data from RAM to application flash(*)
Read RAM data and write to application flash
(3)
Hibernate Mode
(4)
Hibernate Wakeup Reset
Return program
Nomal Operation (Active mode)
(5)
Transfer data from application flash to RAM(*)
Read application flash data and write to RAM
End
Note: (*) Transfer processing needs arbitration with other programs

Figure 15 **Flowchart of example backup procedure in Hibernate mode transition for backup memory data**

1. When in Active mode of normal operation, set Hibernate mode.
2. Transfer the backup memory data from the RAM to application flash (Read from RAM and Write to the application flash)
3. MCU enters Hibernate mode.
4. When a Hibernate wakeup reset occurs, MCU changes from Hibernate mode to Active mode.
5. Program execution resumes. The backup memory data is transferred from the application flash to RAM. (Read to the application flash and Write to the RAM)

Table 9 lists the parameters and Table 10 lists the functions of the configuration part in SDL for Hibernate mode transition settings.

Table 9 **List of initial in Hibernate mode transition for backup memory data parameters**

Parameters	Description	Value
Hibernate_wakeup_port_pin_cfg. outVal	Pin output state	0ul
Hibernate_wakeup_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_HIGHZ
Hibernate_wakeup_port_pin_cfg. hsiom	Connection for I/O pin route	WAKEUP_PIN_MUX
Hibernate_wakeup_port_pin_cfg. intEdge	Edge which will trigger an IRQ	CY_GPIO_INTR_RISING
Hibernate_wakeup_port_pin_cfg. intMask	Masks edge interrupt	1ul
Hibernate_wakeup_port_pin_cfg. vtrip	Input buffer mode	0ul,

(table continues...)

Table 9 (continued) List of initial in Hibernate mode transition for backup memory data parameters

Parameters	Description	Value
Hibernate_wakeup_port_pin_cfg. slewRate	Slew rate	0ul
Hibernate_wakeup_port_pin_cfg. driveSel	GPIO drive strength	0ul
user_led0_port_pin_cfg. outVal	Pin output state	0ul
user_led0_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_STRONG_IN_OFF
user_led0_port_pin_cfg. hsiom	Connection for I/O pin route	USER_LED0_PIN_MUX
user_led0_port_pin_cfg. intEdge	Edge which will trigger an IRQ	0ul
user_led0_port_pin_cfg. intMask	Masks edge interrupt	0ul
user_led0_port_pin_cfg. vtrip	Input buffer mode	0ul
user_led0_port_pin_cfg. slewRate	Slew rate	0ul
user_led0_port_pin_cfg. driveSel	GPIO drive strength	0ul
user_led1_port_pin_cfg. outVal	Pin output state	0ul
user_led1_port_pin_cfg. driveMode	GPIO drive mode	CY_GPIO_DM_STRONG_IN_OFF
user_led1_port_pin_cfg. hsiom	Connection for I/O pin route	USER_LED1_PIN_MUX
user_led1_port_pin_cfg. intEdge	Edge which will trigger an IRQ	0ul
user_led1_port_pin_cfg. intMask	Masks edge interrupt	0ul
user_led1_port_pin_cfg. vtrip	Input buffer mode	0ul
user_led1_port_pin_cfg. slewRate	Slew rate	0ul
user_led1_port_pin_cfg. driveSel	GPIO drive strength	0ul

Table 10 List of initial in Hibernate mode transition for backup memory data functions

Functions	Description	Value
Cy_GPIO_Pin_Init (volatile stc_GPIO_PRT_t *base, uint32_t pinNum, const cy_stc_gpio_pin_config_t *config)	Initializes all pin configuration setting for the pin. *base: Pointer to the pin's port register base address pinNum: Position of the pin bit-field within the port register *config: Pointer to the pin config structure base address	WAKEUP_PORT, WAKEUP_PIN, &Hibernate_wakeup_port_pin_cfg
Cy_SysPm_GetIoFreezeStatus(void)	Checks whether IoFreeze is enabled.	-

(table continues...)

Table 10 (continued) List of initial in Hibernate mode transition for backup memory data functions

Functions	Description	Value
Cy_SysPm_IoUnfreeze(void)	Unfreezes the IOs.	-
Cy_SysReset_IsResetDueToHibWakeup (void)	Returns reset state if it is due to Hibernate wakeup.	-
Cy_FlashInit(bool non_blocking)	Initializes application flash. non_blocking: use non-blocking mode.	false
Cy_FlashWriteWork (uint32_t writeAddr, const uint32_t* data, cy_en_flash_driver_blocking_t blocking)	Programs the 32-bit data to address in the application flash. writeAddr: address of application flash sector to be written into data: pointer to data to be written from. Blocking: param blocking Blocking mode or not	TEST_WF_LS_ADDR, (uint32_t*)&rget_value, CY_FLASH_DRIVER_BLOCKING
Cy_SysPm_SetHibWakeupSource (cy_en_syspm_hib_wakeup_source_t wakeupSource)	Configures sources to wake up the device from the Hibernate mode. wakeupSource: Source to wakeup	CY_SYSPM_HIBPIN0_HIGH
Cy_SysPm_Hibernate(void)	Sets the device into Hibernate mode.	-

Code Listing 17 shows an example program of configuration part for Hibernate mode transition for backup memory data.

Code Listing 17 Example program to Hibernate mode transition for backup memory data function

```
/* Define for hibernate wakeup gpio */
/* Define the wakeup port settings.*/
#define WAKEUP_PORT                     GPIO_PRT21
#define WAKEUP_PIN                      4ul
#define WAKEUP_PIN_MUX                  P21_4_GPIO
#define WAKEUP_IRQ                      ioss_interrupts_gpio_21_IRQn

/* Define the transition data. */
#define RAM_0_DATA                      0x5A5A5A5Aul

/* Access to Application flash address */
/* Define the address for Flash.*/
#define TEST_WF_LS_ADDR                 0x14000010ul

/* Set to wakeup pin for Hibernate mode return */
const cy_stc_gpio_pin_config_t Hibernate_wakeup_port_pin_cfg =
{
  /* Configure the port setting parameters for wakeup. See Table 9.*/
 .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_HIGHZ,
  .hsiom     = WAKEUP_PIN_MUX,
  .intEdge   = CY_GPIO_INTR_RISING,
  .intMask   = 1ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel  = 0ul,
};

cy_stc_gpio_pin_config_t user_led0_port_pin_cfg =
{
/  * Configure the port setting parameters for LED0. See Table 9 .*/
  .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
  .hsiom     = CY_LED0_PIN_MUX,
  .intEdge   = 0ul,
  .intMask   = 0ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel  = 0ul,
};

cy_stc_gpio_pin_config_t user_led1_port_pin_cfg =
{
  /* Configure the port setting parameters for LED1. See Table 9.*/
  .outVal    = 0ul,
  .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
  .hsiom     = CY_LED1_PIN_MUX,
  .intEdge   = 0ul,
  .intMask   = 0ul,
  .vtrip     = 0ul,
  .slewRate  = 0ul,
  .driveSel   = 0ul,
};
/* Set to IRQ for hibernate wakeup gpio */
/* Configure the interrupt structure parameters.*/
const cy_stc_sysint_irq_t hibwakeup_irq_cfg =
{
  .sysIntSrc   = WAKEUP_IRQ,
  .intIdx      = CPUIntIdx2_IRQn,
  .isEnabled   = true,
};
/* Hibernate mode start flag */
uint8_t Hibernate_transition = false;

/* SRAM address for access */
/* Define the address for RAM0.*/
uint32_t variable_ram0_address = 0x08000818ul;

void Wakeup_IntHandler(void)      //(4) Interrupt routine for Hibernate wakeup.
{
  uint32_t intStatus;

  /* If wakeup raising edge detected */
  intStatus = Cy_GPIO_GetInterruptStatusMasked(WAKEUP_PORT, WAKEUP_PIN);
  if (intStatus != 0ul)
  {
    /* Clear to interrupt flag */
    Cy_GPIO_ClearInterrupt(WAKEUP_PORT, WAKEUP_PIN);
  }
}

int main(void)
{

  uint32_t RstReason = 0ul;

  __enable_irq(); /* Enable global interrupts. */

  SystemInit();

  /* Initialize for LED0, LED1 */
  /* Set for LED0, LED1.*/
  Cy_GPIO_Pin_Init(CY_LED0_PORT, CY_LED0_PIN, &user_led0_port_pin_cfg);
  Cy_GPIO_Pin_Init(CY_LED1_PORT, CY_LED1_PIN, &user_led1_port_pin_cfg);


  /* Initialize for wakeup pin(P21_4) */
  /* Set GPIO pin for Hibernate wakeup. See Code Listing 14.*/
  Cy_GPIO_Pin_Init(WAKEUP_PORT, WAKEUP_PIN, &Hibernate_wakeup_port_pin_cfg);

  /* Check I/O frozen status, unfreeze if it was frozen.*/
  if(Cy_SysPm_GetIoFreezeStatus())     //Check if the Frozen status of the I/O port. See Code
Listing 18.
  {
    /* Unfreeze the system */
    /* If it is in freeze status, release it. See Code Listing 19.*/
    Cy_SysPm_IoUnfreeze();
  }

  /* Interrupt settings */
  Cy_SysInt_InitIRQ(&hibwakeup_irq_cfg);
  Cy_SysInt_SetSystemIrqVector(hibwakeup_irq_cfg.sysIntSrc, Wakeup_IntHandler);
  NVIC_SetPriority(hibwakeup_irq_cfg.intIdx, 3ul);
  NVIC_EnableIRQ(hibwakeup_irq_cfg.intIdx);

  /* Check if the Hibernate wakeup reset */
  /* Check if the Hibernate wakeup reset cause in restart. See Code Listing 20.*/
  RstReason = Cy_SysReset_IsResetDueToHibWakeup();

  /* If the cause of RESET was Hibernate Wakeup */
  if(RstReason == true)
  {
    /*_Application flash read data value */
    uint32_t fget_value = 0ul;

    /* Read the data from Application flash(Workflash) */
    /* (5) Read to data in application flash and write to RAM in restart.*/
    fget_value = CY_GET_REG32(TEST_WF_LS_ADDR);

    /* Write the read data to RAM */
    /* (5) Read to data in application flash and write to RAM in restart.*/
    CY_SET_REG32(variable_ram0_address, fget_value);

    /* Check to flash read data */
    if(fget_value == 0x5A5A5A5Aul)
    {
      /* Turn on LED0 to indicate that Hibernate wakeup has occurred */
      for(uint16_t i = 0ul; i < 50ul; i++)     //The LED0 blink several times to indicate
completion of processing after Hibernate wakeup in restart.
      {
        Cy_SysTick_DelayInUs(50000ul);
        Cy_GPIO_Inv(CY_LED0_PORT, CY_LED0_PIN);
      }
    }
  }

  /* Hibernate mode transition start setting */
  /* (1) Set for Hibernate mode transtion.*/
  Hibernate_transition = true;

  /* Prepare the data "0x5A5A5A5A" in RAM0. Write to RAM0 in this data.  */
  CY_SET_REG32(variable_ram0_address, RAM_0_DATA);     //Prepare for transition data.

  for(;;)
  {
    /* Toggle an LED1 to notify MCU is working */
    for(uint16_t i = 0ul; i < 100ul; i++)     //The LED1 blink several times to indicate that
```

```
it is in normal operation.
    {
      Cy_SysTick_DelayInUs(50000ul);
      Cy_GPIO_Inv(CY_LED1_PORT, CY_LED1_PIN);
    }

    /* RAM read data value */
    uint32_t rget_value = 0ul;

    /* Read in the RAM data "0x5A5A5A5A"  */
    rget_value = CY_GET_REG32(variable_ram0_address);



    /* Initialize to Flash */
    /* Initialize to application flash. See Code Listing 21.*/
    Cy_FlashInit(false);



    /* Write to Application flash(Work flash)  */
    /* Write to application flash. See Code Listing 22.*/
    Cy_FlashWriteWork(TEST_WF_LS_ADDR, (uint32_t*)&rget_value,
    CY_FLASH_DRIVER_BLOCKING);

    /* Enable hibernate wake up source pin */
    /* Set for wakeup source. See Code Listing 23.*/
    Cy_SysPm_SetHibWakeupSource(CY_SYSPM_HIBPIN0_HIGH);

    /* Transfer to Hibernate Mode  */
    /* (3) MCU enters Hibernate mode. See Code Listing 24.*/
    Cy_SysPm_Hibernate();

  }
}
```

Code Listing 18 to Code Listing 24 show an example program to configure Hibernate mode transition for backup memory data in the driver part.

Code Listing 18 Example program to GetIoFreezeStatus in driver part

```
__STATIC_INLINE bool Cy_SysPm_GetIoFreezeStatus(void)
{
  return(0u != _FLD2VAL(SRSS_PWR_HIBERNATE_FREEZE, SRSS->unPWR_HIBERNATE.u32Register));
}
```

Code Listing 19 Example program to IoUnfreeze in driver part

```
/*Check if the Frozen status of the I/O port.*/
void Cy_SysPm_IoUnfreeze(void)
{
  uint32_t interruptState;
  interruptState = Cy_SysLib_EnterCriticalSection();

  /* Preserve the last reset reason and wakeup polarity. Then, unfreeze I/O:
  * write PWR_HIBERNATE.FREEZE=0, .UNLOCK=0x3A, .HIBERANTE=0,
  */
  /* Unfreeze I/O port.*/
  SRSS->unPWR_HIBERNATE.u32Register = (SRSS->unPWR_HIBERNATE.u32Register &
  CY_SYSPM_PWR_RETAIN_HIBERNATE_STATUS) | CY_SYSPM_PWR_HIBERNATE_UNLOCK;

  /* If Read stands after Write, read this register two times to delay
  *  enough time for internal settling.
  */
  (void) SRSS->unPWR_HIBERNATE.u32Register;
  (void) SRSS->unPWR_HIBERNATE.u32Register;

  /* Lock the hibernate mode:
  * write PWR_HIBERNATE.HIBERNATE=0, UNLOCK=0x00, HIBERANTE=0
  */
  SRSS->unPWR_HIBERNATE.u32Register &= CY_SYSPM_PWR_RETAIN_HIBERNATE_STATUS;

  Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 20 Example program to IsResetDueToHibWakeup in driver part

```
bool Cy_SysReset_IsResetDueToHibWakeup(void)
{
  bool retReason = false;

  // Contains a 8-bit token that is retained through a HIBERNATE/WAKEUP sequence
  // that can be used by firmware to differentiate WAKEUP from a general RESET event
  // Value 0x1B into the TOKEN is added before jumping to HIBERNATE
  if(0x1Bu == SRSS->unPWR_HIBERNATE.stcField.u8TOKEN)
  {
    retReason = true;    //Check the Hibernate wakeup reset cause
  }

  return retReason;
}
```

Code Listing 21 Example program to FlashInit in driver part

```
void Cy_FlashInit(bool non_blocking)
{
  if(non_blocking == true)
  {
    /******  Setting for IPCs    ******/
    Cy_Srom_SetResponseHandler(Cy_FlashHandler, CPUIntIdx3_IRQn);
    NVIC_SetPriority(CPUIntIdx3_IRQn, 3ul);
    NVIC_EnableIRQ(CPUIntIdx3_IRQn);
    g_NB_ModeEnabled = true;
  }
  else
  {
    g_NB_ModeEnabled = false;
  }

  /*  Flash Write Enable   */
  /* Initialization for writing to application Flash.*/
  Cy_Flashc_MainWriteEnable();
  Cy_Flashc_WorkWriteEnable();

  g_completeFlag = true;
}
```

Code Listing 22 Example program to FlashWriteWork in driver part

```
void Cy_FlashWriteWork(uint32_t writeAddr, const uint32_t* data, cy_en_flash_driver_blocking_t
blocking)
{
  CY_ASSERT(Cy_Flash_WorkBoundsCheck(writeAddr) == CY_FLASH_IN_BOUNDS);

  uint32_t status;
  cy_stc_flash_programrow_config_t programRowConfig = {0};

  if(blocking == CY_FLASH_DRIVER_NON_BLOCKING)
  {
    CY_ASSERT(g_NB_ModeEnabled == true);

    /* Only for non-blocking operation */
    g_completeFlag = false;
  }

  //  Program work flash
  programRowConfig.blocking = CY_FLASH_PROGRAMROW_BLOCKING;
  programRowConfig.skipBC   = CY_FLASH_PROGRAMROW_SKIP_BLANK_CHECK;
  programRowConfig.dataSize = CY_FLASH_PROGRAMROW_DATA_SIZE_32BIT;
  programRowConfig.dataLoc  = CY_FLASH_PROGRAMROW_DATA_LOCATION_SRAM;
  programRowConfig.intrMask = CY_FLASH_PROGRAMROW_NOT_SET_INTR_MASK;
  programRowConfig.destAddr = (uint32_t*)writeAddr;
  programRowConfig.dataAddr = data;
  /* Write to data in application flash.*/
  status = Cy_Flash_ProgramRow(NULL, &programRowConfig, blocking);
  CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
}
```

Code Listing 23 Example program to SetHibWakeupSource in driver part

```
void Cy_SysPm_SetHibWakeupSource(cy_en_syspm_hib_wakeup_source_t wakeupSource)
{
  /* Reconfigure the wake-up pins and LPComp polarity based on the input */
  if(0u != ((uint32_t) wakeupSource & CY_SYSPM_WAKEUP_LPCOMP0))
  {
  SRSS->unPWR_HIBERNATE.u32Register &=
    ((uint32_t) ~_VAL2FLD(SRSS_PWR_HIBERNATE_POLARITY_HIBPIN, CY_SYSPM_WAKEUP_LPCOMP0_BIT));
  }

            if(0u != ((uint32_t) wakeupSource & CY_SYSPM_WAKEUP_LPCOMP1))
  {
  SRSS->unPWR_HIBERNATE.u32Register &=
    ((uint32_t) ~_VAL2FLD(SRSS_PWR_HIBERNATE_POLARITY_HIBPIN, CY_SYSPM_WAKEUP_LPCOMP1_BIT));
  }

            if(0u != ((uint32_t) wakeupSource & CY_SYSPM_WAKEUP_PIN0))
  {
  SRSS->unPWR_HIBERNATE.u32Register &=
    ((uint32_t) ~_VAL2FLD(SRSS_PWR_HIBERNATE_POLARITY_HIBPIN, CY_SYSPM_WAKEUP_PIN0_BIT));
  }

            if(0u != ((uint32_t) wakeupSource & CY_SYSPM_WAKEUP_PIN1))
  {
    SRSS->unPWR_HIBERNATE.u32Register &=
    ((uint32_t) ~_VAL2FLD(SRSS_PWR_HIBERNATE_POLARITY_HIBPIN, CY_SYSPM_WAKEUP_PIN1_BIT));
  }

  /* Set for wakeup source in pin0.*/
  SRSS->unPWR_HIBERNATE.u32Register |= ((uint32_t) wakeupSource);
}
```

Code Listing 24 Example program to SysPm_Hibernate in driver part

```
cy_en_syspm_status_t Cy_SysPm_Hibernate(void)
{
  cy_en_syspm_status_t retVal = CY_SYSPM_SUCCESS;

  /* Call the registered callback functions with the
  * CY_SYSPM_CHECK_READY parameter
  */
  if(0u != currentRegisteredCallbacksNumber)
  {
    retVal = Cy_SysPm_ExecuteCallback(CY_SYSPM_HIBERNATE, CY_SYSPM_CHECK_READY);
  }

  /* The device (core) can switch into Hibernate power mode only when
  *  all executed registered callback functions with CY_SYSPM_CHECK_READY
  *  parameter returned CY_SYSPM_SUCCESS.
  */
  if(retVal == CY_SYSPM_SUCCESS)
  {
    /* Call registered callback functions with CY_SYSPM_BEFORE_TRANSITION
    *  parameter. Return value is ignored.
    */
    (void) Cy_SysLib_EnterCriticalSection();
    if(0u != currentRegisteredCallbacksNumber)
    {
      (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_HIBERNATE, CY_SYSPM_BEFORE_ENTER);
    }

    /* Preserve the token that will retain through a wakeup sequence
    * thus could be used by Cy_SysReset_GetResetReason() to differentiate
    * Wakeup from a general reset event.
    * Preserve the wakeup source(s) configuration.
    */
    /* (3) MCU enters Hibernate mode.*/
    SRSS->unPWR_HIBERNATE.u32Register =
    (SRSS->unPWR_HIBERNATE.u32Register & CY_SYSPM_PWR_WAKEUP_HIB_MASK) |
CY_SYSPM_PWR_TOKEN_HIBERNATE;

    /* All the three writes to hibernate register use the same value:
    *  PWR_HIBERNATE.FREEZE=1, .UNLOCK=0x3A, .HIBERANTE=1,
    */
    SRSS->unPWR_HIBERNATE.u32Register |= CY_SYSPM_PWR_SET_HIBERNATE;

    SRSS->unPWR_HIBERNATE.u32Register |= CY_SYSPM_PWR_SET_HIBERNATE;

    SRSS->unPWR_HIBERNATE.u32Register |= CY_SYSPM_PWR_SET_HIBERNATE;

    /* Wait for transition */
    __WFI();

    /* The callback functions calls with the CY_SYSPM_AFTER_TRANSITION
    * parameter in the hibernate power mode are not applicable as device
```

```
         * wake-up was made on device reboot.
         */

        /* A wakeup from the hibernate is performed by toggling of the wakeup
         * pins, or WDT matches, or Backup domain alarm expires. Depends on what
         * item is configured in the hibernate register. After a wakeup event, a
         * normal Boot procedure occurs.
         * No need to exit from the critical section.
         */
      }
      else
      {
        /* Execute callback functions with the CY_SYSPM_CHECK_FAIL parameter to
         * undo everything done in the callback with the CY_SYSPM_CHECK_READY
         * parameter. The return value is ignored.
         */
        (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_HIBERNATE, CY_SYSPM_CHECK_FAIL);
        retVal = CY_SYSPM_FAIL;
      }
      return retVal;
    }
```

5 Glossary

Table 11 Glossary

Terms	Description
CPU	Central Processing Unit
BOD	Brown-Out Detection. See the "Power Supply and Monitoring" chapter of the architecture TRM for details.
LVD	Low-Voltage Detection. See the "Power Supply and Monitoring" chapter of the architecture TRM for details.
VDDD	Digital power supply. See the "Power Supply and Monitoring" chapter of the architecture TRM for details.
GPIO	General purpose input/output. See the "IO System" chapter of the architecture TRM for details.
ECC	Error Correcting Code
CPUSS	CPU subsystem
MCU	Microcontroller Unit

6 Related documents

The following are the TRAVEO™ T2G family series datasheets and Technical Reference Manuals. Contact Technical Support to obtain these documents.

- Device datasheet
  - CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-24601)
  - CYT3BB/4BB datasheet 32- bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-27763)
- Body controller entry family
  - TRAVEO™ T2G automotive body controller entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9
- Body controller high family
  - TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB
- Cluster 2D family
  - TRAVEO™ T2G automotive cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN (Doc No. 002-25923)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL (Doc No. 002-29584)

7 Other references

Infineon provides the Sample Driver Library (SDL) including startup as sample software to access various peripherals. SDL also serves as a reference, to customers, for drivers that are not covered by the official AUTOSAR products. The SDL cannot be used for production purposes as it does not qualify to any automotive standards. The code snippets in this application note are part of the SDL. Contact Technical Support to obtain the SDL.

Revision history

Document version	Date of release	Description of changes
**	2019-06-03	New application note.
*A	2019-10-24	Added CYT4D Series parts related information in all instances across the document. Added "RAM Retention Procedure in Low-Power Mode".
*B	2020-03-12	Changed parts (from CYT2B/CYT4B/CYT4D Series to CYT2/CYT4 Series) in all instances across the document. Added CYT3 Series parts related information in all instances across the document.
*C	2020-09-16	Updated RAM Retention Procedure Overview: Added "Low-Power Mode (Hibernate Mode) Transition Procedure".
*D	2021-03-08	Updated to Infineon template.
*E	2022-06-03	Updated code examples using SDL.
*F	2023-11-09	Template update; no content updated.

Edition 2023-11-09

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2023 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-agf1681379995630