AN1099
Application Notes

PY32T09x Application Notes

Preface

The PY32T09x series microcontrollers are MCUs featuring a high-performance 32-bit ARM® Cortex®-M0+ core with a wide operating voltage range. They embed up to 256 Kbytes of Flash and up to 32 Kbytes of SRAM, with a maximum operating frequency of 72 MHz. The series includes multiple products in various package types.

This application note will help users understand the considerations for using each module of the PY32T09x and quickly get started with development.

Table 1. Applicable Products

Type	Product Series
Microcontroller Series	PY32T09x

Content

1 SRAM Usage Notes

- It is recommended to set IRAM to the SRAM start address and SRAM capacity of the chip. As shown in Figure 1-1 for an example, when using chips from the PY32T090xB series, the IRAM configuration should be set to the SRAM start address and SRAM capacity of the chip. For the SRAM capacity of the PY32T090 series, please refer to Table 1-1 in the “PY32T090_ Datasheet”:

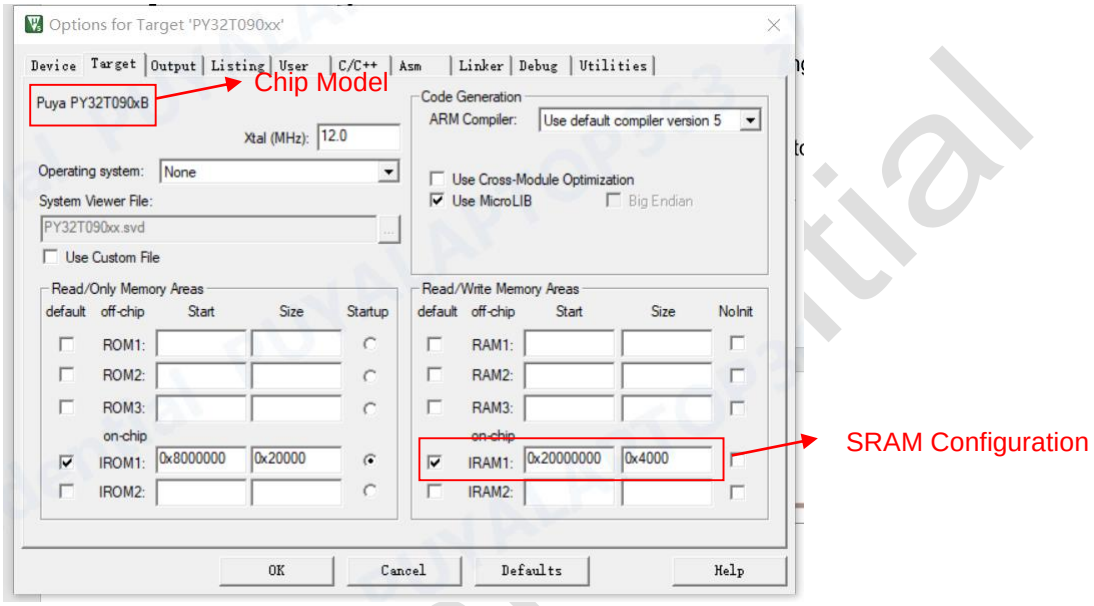

Figure 1-1

- If the configured IRAM size is smaller than the SRAM capacity of the chip, the SRAM space outside the configured IRAM must be manually initialized to zero.

2 PWR Usage Notes

2.1 Precautions for LPRUN/LPSLEEP Mode Operation

- Reset source limitations: only PIN reset and IWDG reset are supported. The following system software resets must not be enabled:
  - Window watchdog reset (WWDG)
  - Option byte loading reset (OBL)
  - SYSRESETREQ software reset
  - SRAM parity error reset
- Before switching to LPRUN/LPSLEEP mode (e.g., RUN/STOP/STANDBY), the following conditions must be met:
  - Do not enable IWDG or manually refresh IWDG_KR_KEY[15:0] to 0xAAAA to ensure IWDG does not generate a reset during switching;
  - Manually clear SYSCFG->SCSR_PERR_RSTEN so that an SRAM parity error generates an NMI interrupt instead of a system reset;

2.2 STOP mode operation notes

- Clock limitation: only 8 MHz HSI is supported to enter STOP mode
- The following conditions must be met before switching modes:

- Manually refresh IWDG_KR_KEY[15:0] to 0xAAAA to ensure IWDG does not generate a reset when switching modes
- Manually refresh WWDG_CR_T[6:0] to the preset counter value to ensure WWDG does not generate a reset when switching modes
- Manually clear SYSCFG_SCSR_PERR_RSTEN so that an SRAM parity error generates an NMI interrupt instead of a system reset
- The reset signal on the NRST pin must be held low for ≥80μs

■ Post-wakeup handling: if the WFE instruction is used to enter STOP mode, insert ≥10 NOPs after the instruction to ensure clock gating synchronization.

3 ADC Usage Notes

3.1 ADC hardware usage notes

■ Clock constraint: AD_CLK frequency ≤ PCLK; PCLK or the ADC asynchronous clock can be selected via the ADC_CCR.CKMODE register

3.2 ADC software usage notes

■ When ADC is configured in discontinuous mode with an injected conversion sequence, software triggering is not allowed; only peripheral triggering is supported, and AUTODLY mode must be disabled. The specific code is as follows:

```
ADC_HandleTypeDef *hadc;

/* ADC Init Code Here*/
/* Discontinuous Conversion */
SET_BIT(hadc->Instance->CFGR, ADC_CFGR_DISCEN);
CLEAR_BIT(hadc->Instance->CFGR, ADC_CFGR_CONT);

/* Disable Autodelay */
CLEAR_BIT(hadc->Instance->CFGR, ADC_CFGR_AUTDLY);

/* Software trigger is forbidden */
SET_BIT(hadc->Instance->JSQR, ADC_CFGR_JEXTEN);
```

■ When the ADC is configured for single conversion mode with an injected conversion sequence, software triggering is forbidden; only peripheral triggering is supported. The specific code is as follows:

```
ADC_HandleTypeDef *hadc;

/* ADC Init Code Here*/
/* Single Conversion */
CLEAR_BIT(hadc->Instance->CFGR, ADC_CFGR_DISCEN);
CLEAR_BIT(hadc->Instance->CFGR, ADC_CFGR_CONT);

/* Software triggering is prohibited */
SET_BIT(hadc->Instance->JSQR, ADC_CFGR_JEXTEN);
```

■ When the ADC is configured for injected conversion with software trigger plus wait mode, it is necessary to wait until the injected channel sequence ends, i.e., ADC_ISR_JEOS = 1, then clear it by software before starting the ADC. The specific code is as follows:

```
ADC_HandleTypeDef *hadc;
ADC_InjectionConfTypeDef *pConfigInjected
```

```
/* Injected Conversion Configure*/
pConfigInjected->ExternalTrigInjecConv = ADC_INJECTED_SOFTWARE_START;

/* Other Injected Conversion Configure Here*/

/* Enable Autodelay */
SET_BIT(hadc->Instance->CFGR, ADC_CFGR_AUTDLY);

While(1)
{
   /* Start ADC Conversion */
   SET_BIT(hadc->Instance->CR, ADC_CR_JADSTART);

/* Wait for JEOS is set, then clear JEOS */
   while(__HAL_ADC_GET_FLAG(hadc, ADC_FLAG_JEOS)) == 0);
   CLEAR_BIT(hadc->Instance->ISR, ADC_FLAG_JEOS);
   /* User Code here */
}
```

4 LPTIM Usage Notes

- Before disabling encoder mode, all flags must be cleared, especially the UP and DOWN flags;
- Steps for configuring the LPTIM operating mode; the following code shows configuration steps for two different kernel clocks:
  - Set the LPTIM kernel clock to PCLK/LSI
  - Enable the clocks of LPTIM1/LPTIM2
  - Configure the LPTIM trigger mode
  - Wait for 5 LPTIM kernel clock cycles, then switch the LPTIM kernel clock to the low-frequency clock
  - Configure single/continuous counting mode. The specific code implementation is as follows:

```
   /* Select PCLK as LPTIM1 Clock source */
   __HAL_RCC_LPTIM1_CONFIG(RCC_LPTIM1CLKSOURCE_PCLK);

   __HAL_RCC_LPTIM1_CLK_ENABLE();
   /* LPTIM Configure Code: */
   /* LPTIM kernel Clock Enable Code Here */
   /* LPTIM Configure Code Here */

   /* If Select PCLK as LPTIM1 Clock source, Delay 5 NOP */
   __NOP();
   __NOP();
   __NOP();
   __NOP();
   __NOP();
```

```
  /* Select LPTIM kernel clock source as LSI */
  __HAL_RCC_LPTIM1_CONFIG(RCC_LPTIM1CLKSOURCE_LSI);

  /* Set LPTIMx Single/Continuous mode */
hlptim->Instance->CR |= LPTIM_CR_CNTSTRT;
```

```
/* Select LSI as LPTIM1 clock source */
__HAL_RCC_LPTIM1_CONFIG(RCC_LPTIM1CLKSOURCE_LSI);
__HAL_RCC_LPTIM1_CLK_ENABLE();

/* LPTIM Configuration Code: */
/* LPTIM Kernel Clock Enable Code Here */
/* LPTIM Configuration Code Here */

/* If LSI is selected as the LPTIM1 clock source, delay 5 LSI clocks */
LPTIM_DelayLSI();
LPTIM_DelayLSI();
LPTIM_DelayLSI();
LPTIM_DelayLSI();
LPTIM_DelayLSI();

  /* Select LPTIM kernel clock source as LSI */
  __HAL_RCC_LPTIM1_CONFIG(RCC_LPTIM1CLKSOURCE_LSI);

  /* Set LPTIMx Single/Continuous mode */
hlptim->Instance->CR |= LPTIM_CR_CNTSTRT;
```

- When writing to the CFGR register, it is recommended to write the configuration value in a single operation; otherwise, there must be at least 3*LPTIM_kernel_CLK between two write operations. Refer to the following code operations:

```
uint32_t tmpcfgr;
/* If LSI is selected as the LPTIM clock source, delay 3 LSI clocks */
/* Example: Set initialization parameters separately */
hlptim->Instance->CFGR |= hlptim->Init.Clock.Source    |
                  hlptim->Init.Clock.Prescaler;
/* Delay */
DelayLSI();
DelayLSI();
DelayLSI();
hlptim->Instance->CFGR |= hlptim->Init.OutputPolarity    |    hlptim->Init.UpdateMode);
/* Delay */
DelayLSI();
DelayLSI();
DelayLSI();
/* Example: Set initialization parameters at once */
hlptim->Instance->CFGR = tmpcfgr;

void DelayLSI()
{
   /* Calculate the value required for a delay of macro-defined (Delay) us */
```

```
    uint32_t    RatioNops = 40 * (SystemCoreClock / 1000000U) / 5;
    for(uint32_t i=0; i<RatioNops;i++)
    {
      __NOP();
    }
  }
```

5 I2C Usage Notes

- Does not support multi-master arbitration
- After entering the I2C interrupt, the I2C interrupt flag must be cleared manually. It is recommended that customers use the interrupt routine provided in the following example for handling:

```
/**
  * @brief This function handles I2C1 interrupt.
  */
void I2C1_IRQHandler(void)
{
  HAL_I2C_IRQHandler(&I2cHandle);
}
```

6 SPI Usage Notes

- When using TI mode, if you need to disable the SPI function, you must also disable TI mode. The specific code implementation is as follows:

```
/* DeInit SPI1 */
if (HAL_SPI_DeInit(&Spi1Handle) != HAL_OK)
{
    APP_ErrorHandler();
}
/* Disable SPI */
CLEAR_BIT(Spi1Handle->Instance->CR1, SPI_CR1_SPE);
CLEAR_BIT(Spi1Handle->Instance->CR2, SPI_CR2_FRF);
```

7 USART Usage Notes

- During automatic baud rate detection, do not clear the USART_CR3_ABREN and USART_CR1_RE flag bits.
- If precise IDLE frame timing is needed, configure the Stop bit to 1.

8 LPUART Usage Notes

- When writing to the CR3 register, it is recommended to write the configuration value in a single operation; otherwise, there must be an interval of 5 * LPUART_kernel_CLK between two write operations. The specific code implementation is as follows:

```
LPUART_HandleTypeDef *hlpuart;
uint32_t tmpcr3;
```

```
  /* Enable the Driver Enable mode by setting the DEM bit in the CR3 register */
  /* If Select LSI as LPUART Clock source, Delay 5 LSI Clock */
  SET_BIT(hlpuart->Instance->CR3, LPUART_CR3_DEM);
  DelayLSI();
  DelayLSI();
  DelayLSI();
  DelayLSI();
  DelayLSI();

  /* Set the Driver Enable polarity */
  MODIFY_REG(hlpuart->Instance->CR3, LPUART_CR3_DEP, Polarity);
  DelayLSI();
  DelayLSI();
  DelayLSI();
  DelayLSI();
  DelayLSI();

  /* Enable Driver Enable mode by setting the DEM bit in the CR3 register and set the Driver
Enable polarity */
  /* at once */
  tmpcr3 |= LPUART_CR3_DEM;
  tmpcr3 |= LPUART_CR3_DEP;
  hlpuart->Instance->CR3 = tmpcr3;
  /**
    * @brief    Delay 1 LSI
    * @retval None
    */
  void DelayLSI()
  {
    /* Calculate the value required for a delay of macro-defined (Delay) us */
    uint32_t    RatioNops = 40 * (SystemCoreClock / 1000000U) / 5;
    for(uint32_t i=0; i<RatioNops;i++)
    {
      __NOP();
    }
  }
```

9 BOR Usage Notes

- When configuring the BOR-related options in User Option1, if BOR_EN is enabled, BOR_LEV must be set to BOR level1 or above. As shown in Figure 9-1, configure the BOR-related options when using PY32CubeProgrammer.

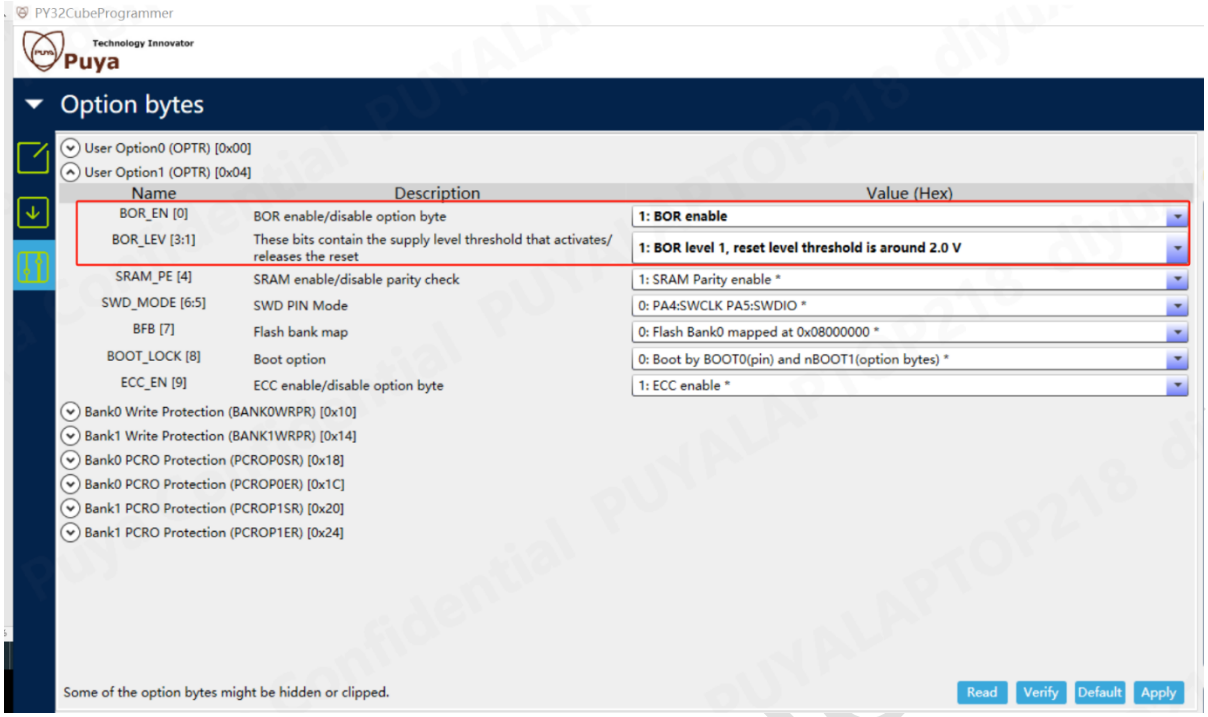

Figure 9-1

10 HSI Usage Notes

- Table 10-1 shows the characteristics of the internal high-speed clock source HSI of the PY32T090. Refer to Table 10-1 and select an appropriate HSI frequency under the corresponding conditions. For more detailed characteristics, refer to Table 5.23 Internal High-Speed Clock Source Characteristics in the "PY32T090_Datasheet".

Table 10-1

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{HSI}	HSI frequency	V_{CC} = 3.3 V, T_A = 25 ℃	-	8.0 16.0 24.0 48.0 64.0	-	MHz
$\Delta_{Temp(HSI)}$	HSI 16/24/48 MHz frequency drift over temperature	V_{CC} = 3.3 V, T_A = 25 ℃	-1(2)	-	1(2)	%
		V_{CC} = 2.0 to 5.5 V, T_A = -20 to 85 ℃	-2(2)	-	2(2)	
		V_{CC} = 1.8 to 5.5 V, T_A = -40 to 105 ℃	-3(2)	-	3(2)	
	HSI 8/64 MHz frequency drift over temperature	V_{CC} = 3.3 V, T_A = 25 ℃	-1(2)	-	1(2)	
		V_{CC} = 2.0 to 5.5 V, T_A = -20 to 105 ℃	-2.5(2)	-	2.5(2)	
		V_{CC} = 1.8 to 5.5 V, T_A = -40 to 105 ℃	-5(2)	-	5(2)	
f_{TRIM}(1)	HSI trimming step	-	0.02	0.1	0.2	%
D_{HSI}(1)	Duty cycle	-	45(1)	-	55(1)	%
$t_{Stab(HSI)}$	HSI stabilization time	-	-	3	4(1)	μs

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$I_{CC(HSI)}$(2)	HSI power consumption	8 MHz	-	108	-	μA
		16 MHz	-	164	-	
		24 MHz	-	221	-	
		48 MHz	-	326	-	
		64 MHz	-	436	-	

11 GPIO Usage Notes

- Before enabling the 120 mA sink current function, ensure the GPIO is configured to output low level; before outputting a high level, this function must be disabled in advance. In the PY32T09x series, the pins with the 120 mA sink current function are PB2~PB9. The following code configures the sink current function of PB2 as an example:

```
GPIO_InitTypeDef    GPIO_InitStruct = {0};
__HAL_RCC_GPIOB_CLK_ENABLE();

/* Initialize GPIO PB2 */
GPIO_InitStruct.Pin = GPIO_PIN_2;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;                /* Push-pull output */
GPIO_InitStruct.Pull = GPIO_PULLUP;                        /* Enable pull-up */
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;         /* GPIO speed */

HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);                    /* GPIO initialization */

/* Set GPIO PB2 Output Low Level*/
GPIOB ->BRR = GPIO_PIN_2;
/* Then Enable LED Pin High Drive*/
SET_BIT (SYSCFG->LEDCFG, SYSCFG_EHS_PB2);

/* To Set GPIO PB2 Output High, LED Pin High Drive Should Be Disabled First*/
CLEAR_BIT (SYSCFG->LEDCFG, SYSCFG_EHS_PB2);
/* Then Set GPIO PB2 Output High Level*/
GPIOB ->BSRR = GPIO_PIN_2;
```

12 Version History

Version	Date	Descriptions
V1.0	2025.01.26	Initial version
V1.1	2025.09.10	1. Added item 9 2. Added item 10 3. Changed chip series name in the document from PY32T090_PY32T092 to PY32T09x
V1.2	2026.03.25	1. Add item 11
V1.3	2026.05.19	1. Modified some register names in Items 3 and 4 2. Updated Table 10-1

Puya Semiconductor Co., Ltd.