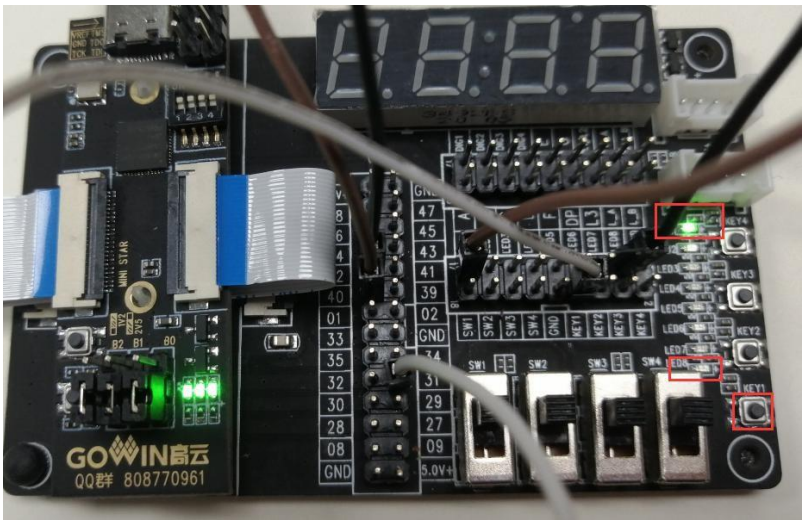
Watchdog Demo

You will learn the basic principle and programming of watchdog by this demo. As shown in the figure below, kick the dog by KEY1. When KEY1 is pressed, LED8 blinks, indicating the kicking dog action; when kicking dog is stopped, it will generate a watchdog interrupt, and the LED1 is always on after the system restart.



This demo includes four parts:

1. Watchdog Introduction
2. Hardware Design
3. Software Design
4. Download and Verification

Watchdog Introduction

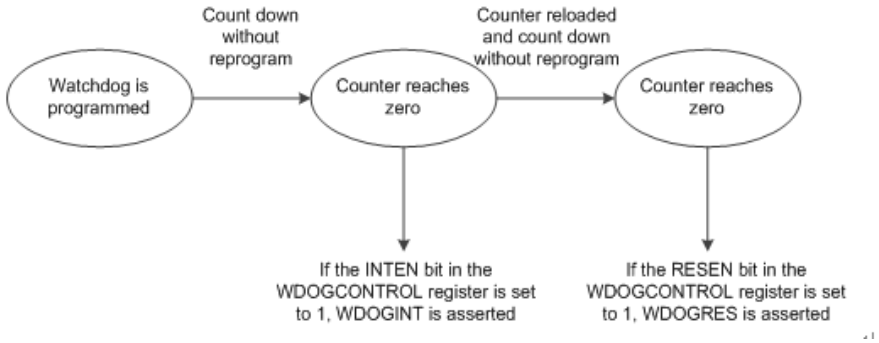
Watchdog is actually a counter; usually given a number, the watchdog starts counting after the program starts running. If the program runs normally, after a while the CPU should issue an instruction that watchdog starts next counting. If the watchdog time out, it is considered that the program is not working normally, and the system is forced to reset.

Therefore, the watchdog is to achieve processor automatic reset (send reset signal) if no kicking signal is received (indicating that the MCU has been hung) within a certain period of time (achieved through the counter).

According to 3.11.9 Watchdog section of DS861, GW1NSR series of FPGA Products Datasheet, you know the development board IP has an embedded Watchdog, which can be controlled and accessed through the APB1 bus. The watchdog is based on a 32 bits down-counter that is initialized from the reload register, WDOGLOAD. The watchdog module generates a regular interrupt, WDOGINT. The counter decrements by one on each positive clock edge of WDOGCLK when the clock enable, WDOGCLKEN, is HIGH. The watchdog monitors the interrupt and asserts a reset request WDOGRES signal when the counter reaches 0. On the next enabled WDOGCLK clock edge, the counter is reloaded from the WDOGLOAD register and the countdown sequence continues. The watchdog module applies a reset to a system in the event of a software failure, providing a way to recover from software crashes. For example, if the interrupt is not cleared by the time the counter next reaches 0, the watchdog module initiates the reset signal.

The following depicts the watchdog operation.

Figure 3-41 Watchdog Operation



Three types of system resets are defined in gw1ns4c_syscon.h.

```
#define SYSCON_RSTINFO_SYSRESETREQ_Pos     0       /* System Reset Request bit position */
##define SYSCON_RSTINFO_WDOGRESETREQ_Pos 1       /* WatchDog Reset Requestt bit position */
#define SYSCON_RSTINFO_LOCKUPRESET_Pos 2       /* Lockup Reset bit position */
```

The watchdog register is as shown below.

Table 3-20 Watchdog Register

Name	Base Offset	Type	Data Width	Reset Value	Description
WDOGLOAD	0x00	Read/ Write	32	0xFFFFFFFF	Watchdog Load Register
WDOGVALUE	0x04	Read only	32	0xFFFFFFFF	Watchdog Value Register
WDOGCONTROL	0x08	Read/ Write	2	0x0	Watchdog Control Register [1]: [0]:
WDOGINTCLR	0x0C	Write only	-	0x-	Watchdog Clear Interrupt Register
WDOGRIS	0x10	Read only	1	0x0	Watchdog Raw Interrupt Status Register
WDOGMIS	0x14	Read only	1	0x0	Watchdog Interrupt Status Register
WDOGLOCK	0xC00	Read/ Write	32	0x0	Watchdog Lock Register
WDOGTCR	0xF00	Read/ Write	1	0x0	Watchdog Integration Test Control Register
WDOGTOP	0xF04	Write only	2	0x0	Watchdog Integration Test Output Set Register

- WDOGLOCK[0]:

  0: unlock Enable write access all register

  1: lock Disable write access register

- WDOGCONTROL:

  00:No action

  01:Interrupt

  10:Reset

The register structure is defined in gw1ns4c_wdog.h.

```
typedef struct
{
  uint32_t WDOG_Reload;         /*reload*/
  WDOGLock_TypeDef WDOG_Lock;     /* write access，0 ： LOCK 1: UNLOCK*/
  WDOGInt_TypeDef WDOG_Int;     /* interrupt enable*/
  WDOGRes_TypeDef WDOG_Res;     /* reset enable*/
  WDOGMode_Typedef WDOG_ITMode;     /* test mode*/
}WDOG_InitTypeDef;
```

Hardware Design

This demo can be modified on the basis of the timer interrupt project. LED is on in LOW and off in HIGH. This demo only needs two LEDs, so only three wires are needed to connect LED1, LED8, KEY1 to FPGA ports, as shown in Figure 1.

Software Design

The software design includes two parts: FPGA internal hardware logic and Cotex-M3 software control code, which can be modified on the basis of the timer interrupt project.

FPGA Internal Logic Design

You do not need to modify the HDL, only need to set GPIO[0], GPIO[1], and GPIO[15] in I/O Constraints. Then click Place & Route to generate the logic file fpga_led.fs.

I/O Constraints

	Port	Direction	Diff Pair	Location	Bank	Exclusive	IO Type
1	gpio_io[0]	inout		40	1	False	LVCMOS33
2	gpio_io[10]	inout		drag or ty...		False	LVCMOS18
3	gpio_io[11]	inout		drag or ty...		False	LVCMOS18
4	gpio_io[12]	inout		drag or ty...		False	LVCMOS18
5	gpio_io[13]	inout		drag or ty...		False	LVCMOS18
6	gpio_io[14]	inout		drag or ty...		False	LVCMOS18
7	gpio_io[15]	inout		31	2	False	LVCMOS33
8	gpio_io[1]	inout		42	1	False	LVCMOS33
9	gpio_io[2]	inout		44	1	False	LVCMOS33
10	gpio_io[3]	inout		46	1	False	LVCMOS33
11	gpio_io[4]	inout		30	2	False	LVCMOS33
12	gpio_io[5]	inout		32	2	False	LVCMOS33
13	gpio_io[6]	inout		35	2	False	LVCMOS33
14	gpio_io[7]	inout		33	2	False	LVCMOS33
15	gpio_io[8]	inout		drag or ty...		False	LVCMOS18
16	gpio_io[9]	inout		drag or ty...		False	LVCMOS18
17	reset_n	input		20	3	False	LVCMOS18

Cotex-M3 Software Control Design

This design can be modified on the basis of timer interrupt project. Open led.uvprojx in the Keil_led\PROJECT folder.

1. Group interrupt

   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);

2. Set GPIO0[0] and GPIO0[1] to output to drive LED1 and LED8; set GPIO0[15] to input to identify KEY1 status.

   GPIO0->OUTENSET = 0x00ff; //IO[15:8]: input IO[7:0]: output

3. Initialize watchdog

```
void watchdog_init(unsigned int cycle, int type)
{
  WDOG_UnlockWriteAccess(); //unlock write access
  WDOG_RestartCounter(cycle); //restart counter
  if (type==0) //interrupt type
  {
  WDOG->CTRL = 0; //disable watchdog
  }
  else if (type==1) //interrupt type
  {
  WDOG_SetIntEnable(); //interrupt enable
  }
  else
  {
  WDOG_SetResetEnable(); //reset enable
  WDOG_SetIntEnable(); //interrupt enable
  }
  WDOG_LockWriteAccess(); //lock write access
}
```

4. The main design is as shown below: when reset watchdog, light up LED1; when kick watchdog, light up LED8.

```
int main(void)
{
  SystemInit();
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
  GPIOInit();
  GPIO_SetBit(GPIO0,GPIO_Pin_0); //off
  //whether this is watchdog reset
  if ((SYSCON_GetRstinfoWdogresetreq()) !=0)
  {
    GPIO_ResetBit(GPIO0,GPIO_Pin_0); //on
    SYSCON->RSTINFO = SYSCON_RSTINFO_WDOGRESETREQ;//clear flag
  }
  else
  {
  GPIO_SetBit(GPIO0,GPIO_Pin_0); //off
  }
  GPIO_SetBit(GPIO0,GPIO_Pin_1); //off
  watchdog_init(99999999,2); //open watchdog timer
  while(1)
  {
    if( 0x0000 == (GPIO0->DATA & 0x8000) ) //GPIO[15]==0
    {
       GPIO_ResetBit(GPIO0,GPIO_Pin_1); //on
       Delay(3333000);
       GPIO_SetBit(GPIO0,GPIO_Pin_1); //off
       watchdog_init(99999999,2); //feed wdog
       while(0x0000 == (GPIO0->DATA & 0x8000));
    }
  }
}
```

5. After bulid, the download file led.bin is generated.

Download and Verification

Use Gowin Software to download. The FPGA hardware platform file is fpga_led.fs, and the Cotex-M3 software file is led.bin, so be careful to choose the correct file path and bulid file, and you can see demo running in the video.