Watchdog (WDT)

Introduction to the results of routine experiments

1. What is a watchdog?

Watchdog Timer is a hardware timer in embedded systems, mainly used for system self-detection and recovery. It is like a "supervisor" that continuously monitors the system operation status through the timer and automatically triggers system reset when the system is abnormal.

2. Working principle of watchdog

The working principle of the watchdog includes the following key links:

- Initialization: Set up the watchdog and specify the timeout
- Operation monitoring: "Feed the dog" operation resets the timer periodically
- Reset trigger: If the timeout is exceeded and the dog is not fed, the system will automatically restart

3. Main Application

- Detect and prevent system freezes
- Improve system operation reliability
- Provides automatic recovery mechanism
- Reduce the need for human intervention

Routine code

The example code of this section is located in [Source Code Summary/02.Basic/08.wdt.py]

Complete code

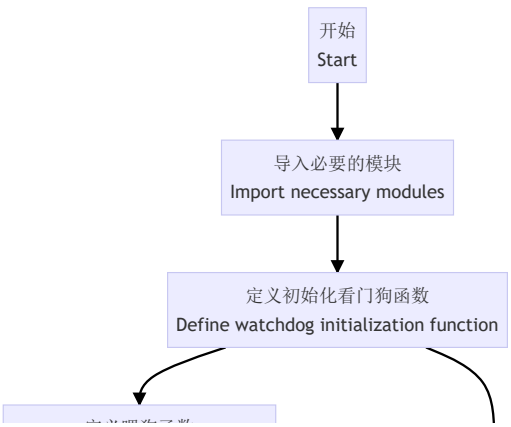
- Note: Please refer to the contents of [Source Code/ 02.Basic / 08.wdt.py] file

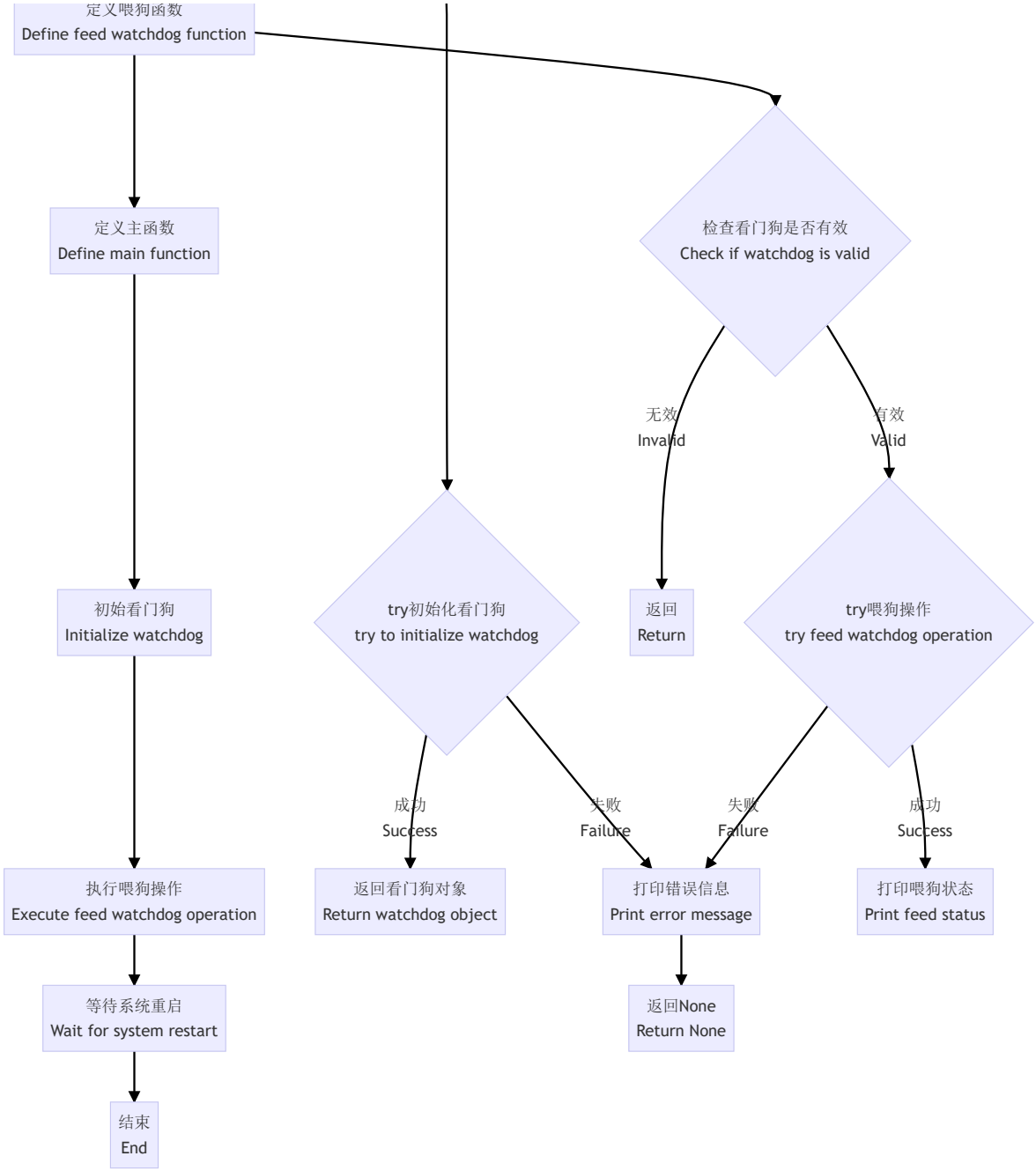
```
# 导入必要的模块 | Import required modules
from machine import WDT  # 导入看门狗模块 | Import watchdog module
import time  # 导入时间模块 | Import time module

"""
详细解析 | Detailed Analysis:
```

```
WDT (Watchdog Timer) 是一种硬件定时器，如果在规定时间内没有被复位（喂狗），
系统就会自动重启。主要用于检测和恢复系统故障。

WDT is a hardware timer that will automatically restart the system if not reset
("fed") within a specified time. It's mainly used for detecting and recovering
from system failures.
"""

def init_watchdog(id=1, timeout=3):
    """初始化看门狗 | Initialize watchdog
    Args:
        id: 看门狗ID | Watchdog ID
        timeout: 超时时间(秒) | Timeout period(seconds)
    Returns:
        WDT对象 | WDT object

    详细解析 | Analysis:
    创建WDT对象，设置超时时间。如果超过timeout秒没有喂狗，系统将重启。
    Creates WDT object and sets timeout. System will restart if not fed within
timeout seconds.
    """
    try:
        return WDT(id, timeout)
    except Exception as e:
        print(f"看门狗初始化失败 | Watchdog initialization failed: {e}")
        return None

def feed_watchdog(wdt, feed_times=3, interval=1):
    """定时喂狗 | Feed watchdog periodically
    Args:
        wdt: 看门狗对象 | Watchdog object
        feed_times: 喂狗次数 | Number of feeds
        interval: 喂狗间隔(秒) | Interval between feeds(seconds)

    详细解析 | Analysis:
    按照指定间隔喂狗指定次数。这个过程会持续 feed_times * interval 秒。
    Feeds the watchdog specified times at given intervals. This process will
last
    for feed_times * interval seconds.
    """
    if not wdt:
        return

    try:
        for i in range(feed_times):
            time.sleep(interval)  # 等待指定时间 | Wait for specified interval
            wdt.feed()  # 喂狗操作 | Feed the watchdog
            print(f"第{i+1}次喂狗 | Feed watchdog {i+1} times")
    except Exception as e:
        print(f"喂狗过程出错 | Error during feeding: {e}")

def main():
    """主函数 | Main function

    详细解析 | Analysis:
    程序的主要执行流程：
    1. 初始化看门狗
    2. 执行定时喂狗
```

```
    3. 停止喂狗等待系统重启

    Main program flow:
    1. Initialize watchdog
    2. Perform periodic feeding
    3. Stop feeding and wait for system restart
    """
    # 初始化看门狗 | Initialize watchdog
    wdt = init_watchdog()
    if not wdt:
        return

    # 喂狗循环 | Feeding loop
    feed_watchdog(wdt)

    # 等待重启 | Wait for restart
    print("停止喂狗，等待系统重启... | Stop feeding, waiting for system restart...")
    while True:
        time.sleep(0.1)  # 降低CPU占用 | Reduce CPU usage

if __name__ == '__main__':
    main()
```

Routine code flow chart




定义喂狗函数
Define feed watchdog function
定义主函数
Define main function
初始看门狗
Initialize watchdog
执行喂狗操作
Execute feed watchdog operation
等待系统重启
Wait for system restart
结束
End
try初始化看门狗
try to initialize watchdog
成功
Success
返回看门狗对象
Return watchdog object
失败
Failure
打印错误信息
Print error message
返回None
Return None
检查看门狗是否有效
Check if watchdog is valid
无效
Invalid
返回
Return
有效
Valid
try喂狗操作
try feed watchdog operation
失败
Failure
成功
Success
打印喂狗状态
Print feed status