2024

Software Architecture & Design

SOLUTION: EXERCISE 2

CALEB GIMPEL

1 Block Diagram

The following is a block diagram of a dual-processor watchdog system. The main processor is a Zynq device running a Linux operating system (OS), such as Petalinux, and hosts a client application and watchdog driver.

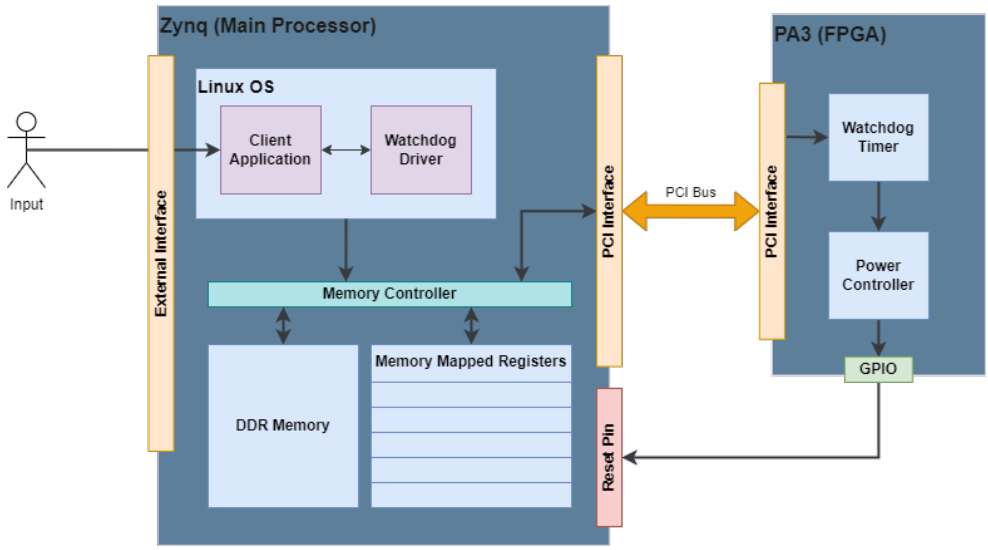

Figure 1 Block diagram of a dual-processor watchdog system.

An assumption for this assignment is that the user has some way to externally connect to the Zynq, which could be Ethernet, UART, JTAG, or otherwise. The discussion of the external interface is beyond the scope of this assignment.

When a user connects to the client application, they can configure a period to “kick” the separate FPGA device called PA3. If a kick is missed due to the OS hanging or the application quitting due to a crash, the PA3 will raise a GPIO connected to the reset pin on the Zynq device.

The Zynq and PA3 are connected over a peripheral component interconnect (PCI) bus and can communicate using memory mapped registers located in the Zynq; the required communication registers are discussed more in section 2.1.

2 State Machine

The flow chart in Figure 2 shows the logical checks made by the PA3 after the user starts the watchdog service. The PA3 first reads the configuration register, then starts the timer countdown and finally runs through its state checks.

There are three signals that are checked to determine the active state: the exit signal, the kick signal, and the timer countdown. The exit signal and kick signal are both provided by the Zynq device and received as a communication register, discussed in section 2.1. The timer countdown is configured at the start of the watchdog service as the number of clock cycles to wait for a kick and located on the PA3.

When the countdown reaches zero remaining clock cycles, the PA3 will trigger a reset by raising a GPIO connected to the Zynq reset pin HIGH. After a reset, the watchdog may perform recovery actions, such as resetting the system state, raising error flags, or logging events. When the Zynq has been fully reset, the PA3 will restart by reading the configuration flags again and starting the timer countdown.

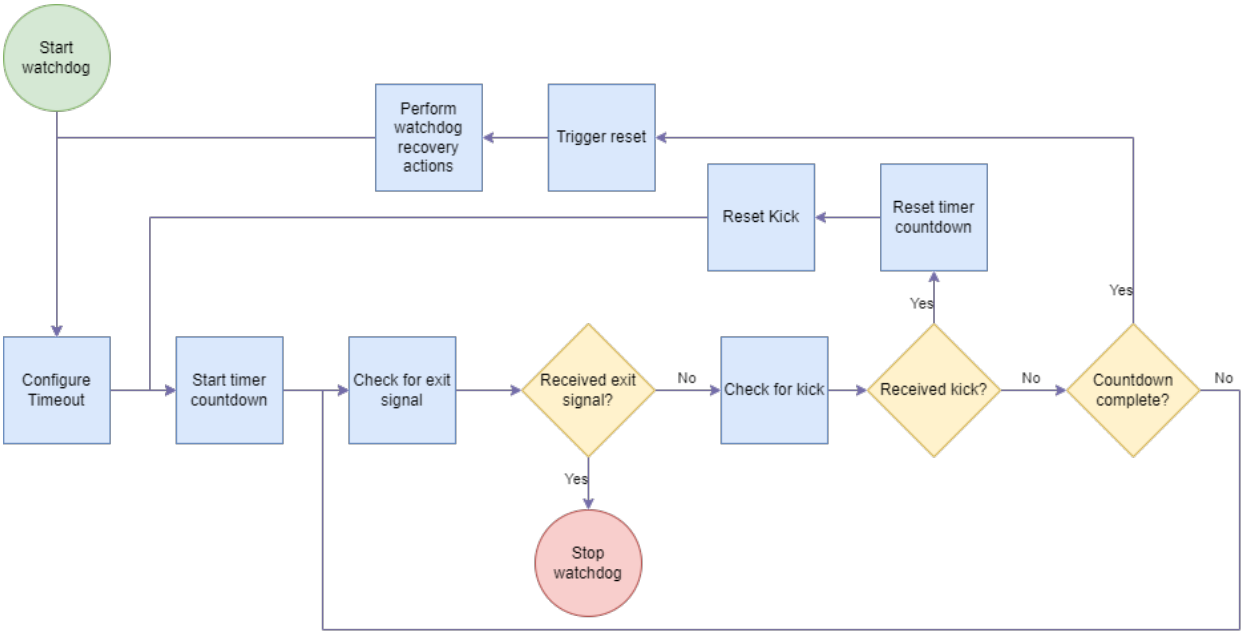

Figure 2 Flow chart of the watchdog service.

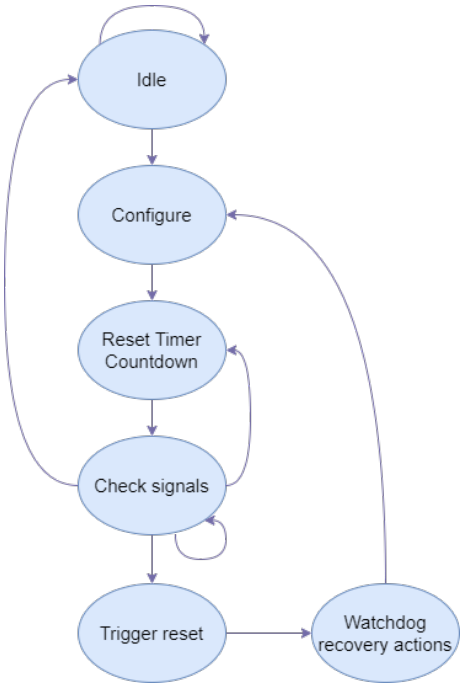

Figure 3 State machine of the watchdog PA3 implementation.

2.1 Communication Registers

The communication registers are memory mapped registers located in the Zynq memory space, as shown in Figure 1. The watchdog service requires just two registers: a configuration and flag register. The register contains information about multiple configurations based on the position of the bits. The following tables include the name of the parameter, the location in the register, and a brief description.

An assumption made for this exercise is that a register can hold a maximum of 32 bits.

Table 1 Configuration Register

Name	Location	Description
Config_period_kick_ms	0xXXXX_FFFF[1]	The period in milliseconds in which the Zynq will send a kick. Maximum: 65.535 seconds.
Config_period_timeout_ms	0xFFFF_XXXX	The period in milliseconds in which to wait for a kick before timing out. Maximum: 65.535 seconds.

[1] Note: There should be some error handling to verify the kick occurs more frequently than the timeout.

Table 2 Flag Register

Name	Location	Description
Flag_enable	0x0000_XXX1	A flag to enable/disable the watchdog timer on the PA3.
Flag_kick	0x0000_XX1X	A flag raised HIGH by the Zynq device, indicating to the PA3 that it is still alive. The flag is reset LOW when the PA3 acknowledges the flag.
Flag_exit	0x0000_X1XX	A flag raised HIGH by the Zynq device, indicating the user would like to gracefully stop the watchdog service.
RESERVED	0x0000_1XXX	A flag bit reserved for future use.

The enable flag is used to enable or disable the watchdog timer functionality altogether. Disabling the watchdog timer can be useful during watchdog recovery actions, system initialization or system operations that require uninterrupted execution.

3 API

The following sections include the commands required to interact with the watchdog driver.

The means of sending the commands through some external interface to the client application are outside the scope of this exercise. For the sake of the example, an assumption is made that the client application can receive and parse a command string and its following arguments.

3.1 Start

The “start” command string will open a connection to the watchdog driver, write the configuration values to the configuration register and begin kicking the control register.

Arguments

Table 3 Client Application Commands

Argument Flag	Argument Type	Range	Default
-k, --kick	int	1-4,095	100
-t, --timeout	Int	1-65,535	1000

Example

```
$ ./watchdog start
> Starting watchdog with kick period 0.100s and timeout period 1.000s
$./watchdog start -k 555 --timeout 12345
> Starting watchdog with kick period 0.555s and timeout period 12.345s.
```

3.2 Stop

The “stop” command string will write a 1 to the flag register, triggering the graceful exit process for the watchdog service.

Example

```
$ ./watchdog stop
> Stopping the watchdog.
```

3.3 Status

The “status” command string will read the flag register to check the enable bit.

Example

```
$ ./watchdog status
> Watchdog is enabled (1).
$ ./watchdog status
> Watchdog is disabled (0).
```

3.4 Enable

The “enable” command string will write a 1 to the enable bit in the flag register.

Example

```
$ ./watchdog enable
> Watchdog is enabled(1)
```

3.5 Disable

The “disable” command string will write a 0 to the enable bit in the flag register.

Example

```
$ ./watchdog disable

> Watchdog is disabled (0)
```

4 Implementation

To implement the software architecture outlined in the previous sections, the following tasks are created with their estimated execution time.

Table 4 Implementation estimate

Task Name	Estimated Time (days)
Watchdog Timer Logic (PA3)	3
Watchdog Driver (Zynq)	3
Client Application API (Zynq)	1
Integration Testing	1
Total	8