Learning Objectives:

PROJECT 3

- Gain experience with threading, synchronization and interprocess communication (IPC)
- Improve the robustness of a software application by making use of a Watchdog timer
- Gain experience with the Xilkernel, a small, highly-configurable (mostly) POSIX-compliant real-time kernel
- Lay the groundwork for networking, file systems, and other OS-dependent functionality for possible use in your final project

There is no demo for Project 3
Project 3 Deliverables due to D2L by 10:00 PM on Sunday 28-Feb-2016

Project Notes:

1. This project is targeted to the Digilent Nexys 4 DDR board. The Nexys 4 board should also work since we are not going to implement an external memory.
2. You may collaborate with your Project 2 partner on this assignment or you may submit your own solution. However, please don't get together with your friends and classmates to do the project. The work you submit must be your own or a joint effort of your team.
3. If you collaborate with your project partner you only need to submit one solution set. Be sure to list all of those who you collaborated with in the header file(s) for the project. Each of you should submit your own set of deliverables if you complete the assignment independently.

4. **IMPORTANT**: Please structure your code cleanly and add meaningful comments so that it is easy to read/grade your code. There is no demo for this project so readability and understandability are of utmost importance.

Project: Real-time Kernel Programming

We will be implementing a robust real-time system that reads the buttons and some of the switches on the Nexys 4 and displays them on the LEDS. You will use the USB serial port on the Nexys 4 to display debug, error and startup messages on a PC. The real-time support will be provided by the Xilkernel. The robustness of the application is achieved by including a watchdog timer that forces a system restart if the timer is not periodically restarted by the application. The robustness can be enhanced further by making use of the Xilkernel and Xilinx API return status and by taking advantage of the system-wide *errno* mechanism supported in Xilkernel.

You will be provided with a starter application (a template, really - it doesn't do anything on its own). The rest is up to you.

Surely an application this simple does not need the overhead and complexity of a priority-based scheduler, interprocess communication (IPC) and synchronization? Of course not, but the adventure is in the learning and the learning is in the doing so go for it.

Target System

The application will be written to run on a target platform that includes the following hardware:

- MicroBlaze, mdm, interrupt controller, etc.
- Local (BRAM) memory **128KB** (The sample application fits in 64K but that is before you add your code)

- Two (2) 32-bit *AXI Timer/Counter*s (each only needs 1 timer, not two) used as follows:
  - Timer 0 – provides the timer tick for the Xilkernel
  - Timer 1 – is not used in the base project but is available for whatever enhancements you implement.
- One (1) *AXI Watchdog Timer* used to provide a Watchdog timer for the system. Configure the Watchdog timer interval to about 2 or 3 seconds lest you get tired of waiting for disaster to strike.
- One (1) *AXI GPIO* with interrupt support. This port connects to the pushbuttons on the Nexys 4. Be sure to configure the peripheral to use interrupts
- One (1) *AXI GPIO*. This port connects to the slide switches on the Nexys 4.
- One (1) *AXI GPIO*. This port connects to LEDs on the Nexys 4.
- One *AXI Interrupt Controller* with the pushbutton *AXI GPIO* interrupt, Timer 0 interrupt, Timer 1 interrupt, and Watchdog timer interrupt as interrupt sources
- One (1) *AXI UART Lite*. This peripheral connects to the USB serial port on the Nexys 4 board. Choose a baud rate that is supported by the terminal emulator (hyperterminal, puTTY, etc.) you use.
- [OPTIONAL] One (1) *AXI Ethernet Lite* used to provide a 10/100 Ethernet link to the system. This peripheral is not required for Project 3, but, you may want to experiment with networking and the LWIP stack.

This is the largest system that you have built for this course but it all fits easily and since there is no external logic you may implement the whole system using an embedded system generated by the IP Integrator with a wrapper file, or using `n4fpga.v` like you did with Project 1. The choice is yours.

NOTE: THIS SYSTEM DOES NOT REQUIRE NEXYS4IO OR PMOD544IO. THERE IS NO ROTARY ENCODER OR DISPLAY AND THE INPUT AND OUTPUT IS THROUGH AXI GPIOS. YOU MAY INSTANTIATE NEXYS4IO AND/OR PMOD544IO IF THEY ARE REQUIRED FOR ANY ENHANCEMENTS YOU INTEND TO IMPLEMENT, BUT KEEP IN MIND THAT THE DRIVERS WERE NOT WRITTEN TO BE REENTRANT.

User Interface

The user interface for the starter application is straightforward. Press a button or switch and display the new button/switch state on the LEDs. The following switches and pushbuttons have dedicated functions:

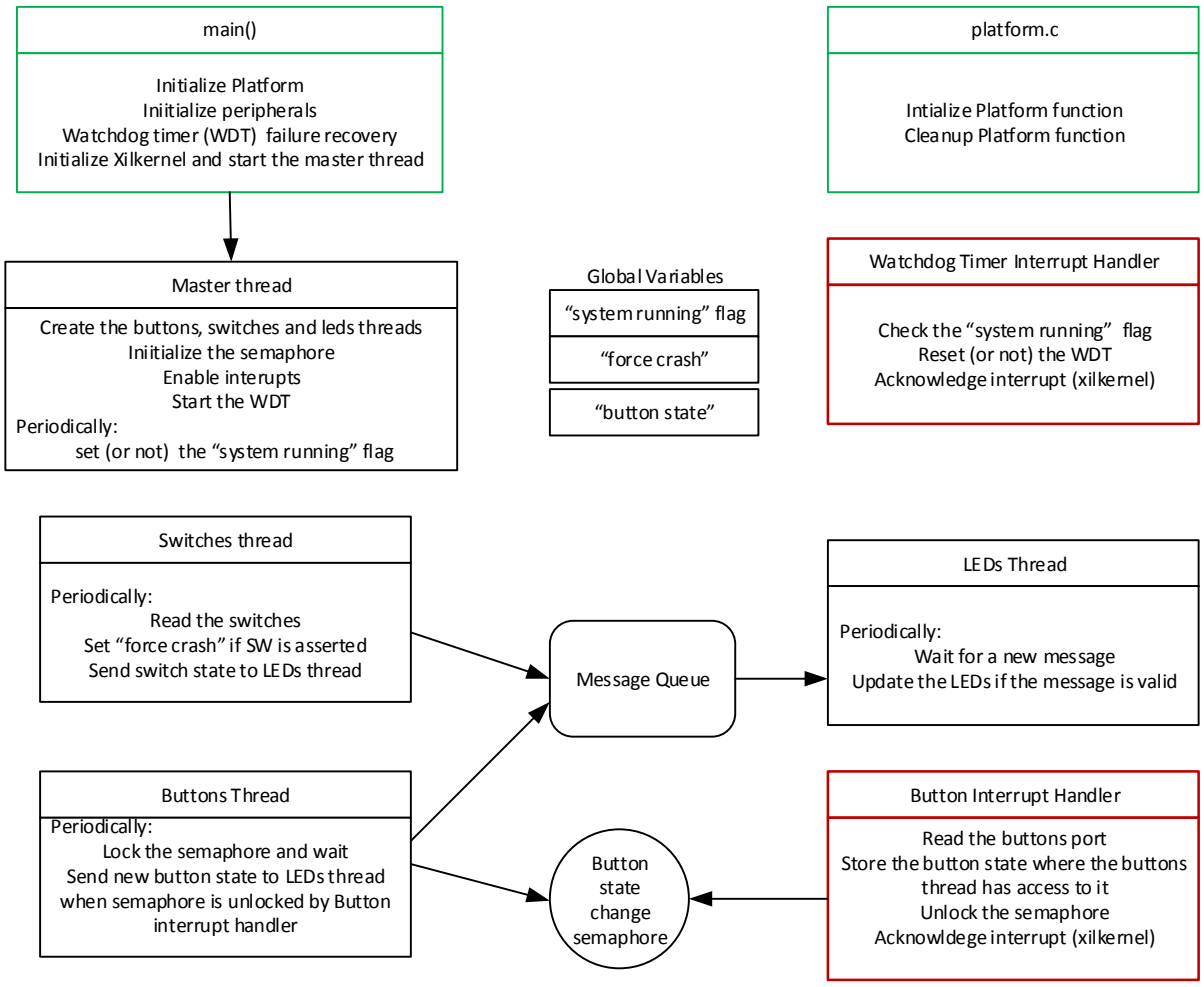
- SW[15] (leftmost) — Turn on (up) to force a Watchdog timer crash. Off (down) for normal operation
- btnCPUReset — Press to force a system reset (asserted low).

The LEDs should be configured the following way:

- LEDS[15] — "Force Crash" switch state
- LEDS[14] — *RESERVED* (can be left off)
- LEDS[13:9] — Pushbutton state
- LEDS[8] — *RESERVED* (can be left off)
- LEDS[7:0] — Switch state for slide switches[7:0] (rightmost)

Suggested Threading Model

While you can choose to implement whatever threading model you think will work best for your application, the starter application is designed with the following threading model in mind:



Project Tasks

Step 1 (10 points) Build the target system

The first part of the assignment is to build the target system described earlier. This can be done pretty easily (famous last words) in Vivado/IP Integrator. Add the correct Nexys 4 constraints (.xdc) file to your project and make any necessary edits. You can start with the `n4fpga.xdc` file you used in your earlier projects.

NOTE: BE SURE TO CHECK THAT THE .XDC FILE YOU ADDED TO THE PROJECT CONTAINS PIN CONSTRAINTS FOR ALL OF THE EXTERNAL PORTS IN YOUR DESIGN. DON'T FORGET TO MATCH THE PORT NAMES IN YOUR TOP LEVEL MODULE WITH THE PORT NAMES IN THE .XDC FILE. DON'T FORGET TO COMMENT OUT (# IN COLUMN 1) ANY `SET_PROPERTY` LINES IN THE .XDC FILE THAT AREN'T USED IN THIS DESIGN (LIKE THE SEVEN SEGMENT DISPLAY SIGNALS, FOR EXAMPLE).

Step 2 (30 points) Implement the watchdog timer functionality

The system described above includes a Xilinx *AXI Watchdog timer* peripheral. This peripheral serves as both a timebase (a simple counter like the FIT timer) and a watchdog timer. Step 2 is to include watchdog timer functionality in the application. We described the operation of the watchdog timer in class. The application and watchdog timer should operate with the following characteristics:

- The application should provide some kind of recovery action. This could be as simple as displaying a message and waiting in an infinite loop for a reset or power cycle, or it could include provision to restart the application. Your application can use the ***`XWdtTb_IsWdtExpired()`*** API function to determine whether a reset was caused by a WDT timeout.
- The application should have the ability to force a WDT crash. This is most easily accomplished by preventing the calls to the ***`XWdtTb_RestartWdt()`*** from occurring when the "Force Crash" switch is on. Use the leftmost slide switch (sw[15]) for the "Force Crash" switch.
- The application should check that the application is running. A simple way to do this is to periodically set a "system running" flag in the master thread. The WDT interrupt handler can read and clear this flag and restart the WDT when the handler is triggered on the first WDT overflow. It would be straightforward to provide better coverage by using individual flags for each of the threads. The flags could be set periodically by each of the tasks and then checked/cleared by the WDT functionality in the master thread. While this would work quite nicely for the LEDs and switches thread, it could be problematic in the buttons thread which is waiting for a change in the button state (signaled by the semaphore) that may not occur often enough to avoid a WDT timeout. Xilkernel provides timed wait semaphores and the `IPC_NOWAIT` flag if you are interested in implementing checking in the buttons thread.
- The WDT should force a CPU reset when it expires the second time (the first generates an interrupt – this is how the WDT operates). To do this the ***WDT_Reset*** output from *AXI Watchdog Timer* peripheral should be connected to the ***Aux_Reset_In*** input on the *proc_sys_reset* block.

Step 3 (50 points) Build the application

Implement the application as described. You can define your own tasking and synchronization scheme for the application if you'd like, but the starter application is written with the threading and synchronization model shown earlier in the document. If you create your own tasking and synchronization scheme be sure to include, at a minimum, a semaphore and a message queue; and please include a short write-up describing the threading model you implemented, explaining why you chose that model instead of the one presented.

The Xilkernel environment is created by making a new board support package for your system in the SDK (*File/New/Xilinx Board Support Package*). Select the *xilkernel* environment instead of the *standalone* platform type. The Xilkernel is configured by selecting the *Xilkernel board support package,* opening the *system.mss* file and clicking on *modify this BSP Settings*.

The configuration used by the starter application (taken from the .mss file) is:

```
BEGIN OS
 PARAMETER OS_NAME = xilkernel
```

```
 PARAMETER OS_VER = 6.1
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER config_bufmalloc = true
 PARAMETER config_debug_support = true
 PARAMETER config_msgq = true
 PARAMETER config_named_sema = true
 PARAMETER config_pthread_mutex = true
 PARAMETER config_sema = true
 PARAMETER config_shm = true
 PARAMETER config_time = true
 PARAMETER mem_table = ((8,20))
 PARAMETER n_prio = 5
 PARAMETER sched_type = SCHED_PRIO
 PARAMETER stdin = axi_uartlite_0
 PARAMETER stdout = axi_uartlite_0
 PARAMETER sysintc_spec = microblaze_0_axi_intc
 PARAMETER systmr_dev = systick_timer
END
```

NOTE: SEVERAL OF THESE PARAMETERS (EX: SYSTMR_DEV = SYSTICK_TIMER) HAVE NAMES TAKEN FROM OUR SPECIFIC SYSTEM CONFIGURATION AND MAY BE DIFFERENT FOR YOUR SYSTEM.

Step 4 (10 points) Provide evidence that you have completed the assignment

We want to see source code of your application, a block diagram of your embedded system and your *system.mss* file. We would also like you to submit a few photos or a short video demonstrating that your application works. Include both the WDT functionality and the LEDs display functionality in the demo. We do not need to see a transcript of messages that your application sends through the serial port unless they are important to your demonstration.

If you complete any of the extra credit or if you enhance your application beyond the assignment then please include a short write-up describing the additional features.

Submit your deliverables to D2L as you normally do. The dropbox for the assignment has been set up using your **Project 2 groups**. If you each decide to submit your own solution please submit your deliverables to the group dropbox (one .zip or .rar file per person, please)

Grading

You will be graded on the following:

- Successfully building the embedded system. (10 pts)
- Successfully integrating the WDT functionality into the application (30 pts)
- Successfully demonstrating (e.g. with a short video and/or pictures) the software application (40 pts)
- The quality of your source code. The better we understand your code without a large amount of effort, the better grade we can give it. Use meaningful variable names; make the headers and comments reflect what is actually happening in your application; structure your code and make it readable through consistent indentation and well defined function interfaces (20 pts)

Extra Credit Opportunities (up to 5 points)

You may do one or more of the following tasks for extra credit:

- Implement code to network-enable this application. For example use the lwIP stack API to establish a telnet session with the PC host and transfer/display the led status on the console.

- Add support for the rotary encoder through the use of a custom peripheral that you develop. Write a simple device driver for your new peripheral. Add additional pthreads, synchronization and/or interprocessor communication to integrate the rotary encoder into the design.

- Enhance the application as you see fit. The enhancements you made should be included in your demonstration pictures and/or video, of if that is not practical, than in a session log that demonstrates the additional functionality.

References

[1] *Xilinx UG646 (Ver 6.2)*, Copyright Xilinx Corporation, April 1, 2015.