**Intro to FreeRTOS**

Benjamin A. Blouin

*benjamin.blouin@temple.edu*

**Summary**

The Xilinx Vivado SDK has incorporated FreeRTOS, an open-source real-time operating system (RTOS). A real-time service provides a transformation of inputs to outputs in an embedded system. The main difference between a traditional OS and an RTOS is the general assumption that a service will be completed within a deterministic-derived hard-time-frame. If a system can not complete its task, it is not suitable for use; types of systems that require a hard-time window usually safety oriented, like aircraft controls, or security systems that check-in at given intervals. This lab explores creating a block design for the Zybo embedded system development board, creating an SDK project using FreeRTOS, and exploring the function libraries that run the RTOS system.

**Introduction**

RTOS has a set of assumptions and constraints allowing for the formulation of a deterministic model (Liu & Layland, 1973; Siewert Sam., 2016):

A1: All services requested on periodic basis, the period is constant

A2: Completion-time < period

A3: Service requests are independent (no known phasing)

A4: Runtime is known and deterministic (Worst-Case Execution Time (WCET) may be used)

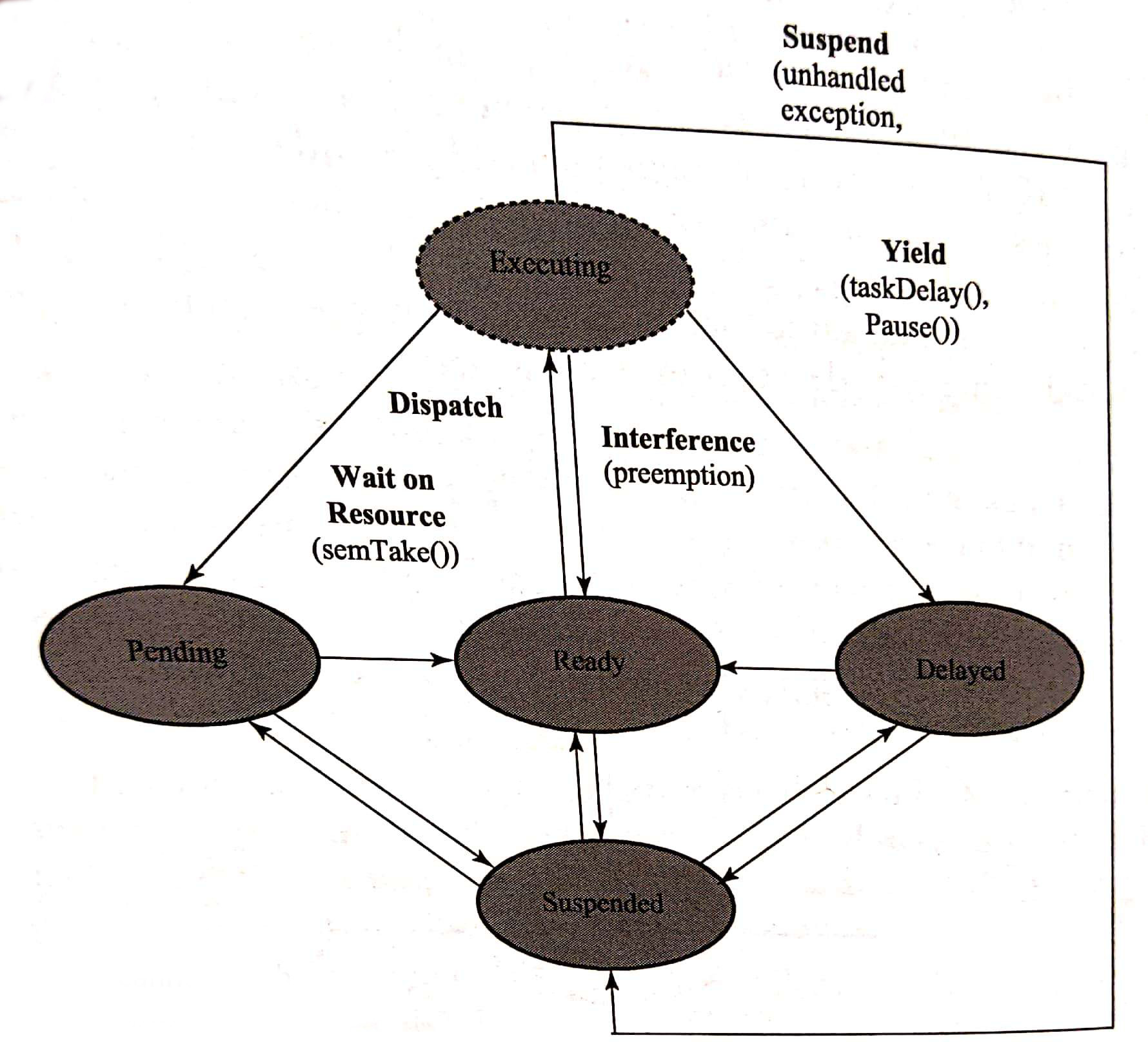
C1: Deadline = period by definition

C2: Fixed-priority, preemptive, run-to-completion scheduling

A5: Critical instant–longest response time for a service occurs when all system services are requested simultaneously (maximum interference case for lowest priority service)

⁠In non-preemptive scheduling policies, like First Come First Serve (FCFS) and Shortest Job Next (SJN), it’s hard to be have hard real-time utility because these services are generally non-deterministic. For hard real-time systems preemptive scheduling allows deterministic, asynchronous state flow. Some preemptive scheduling policies include: Rate Monotonic (RM), Deadline Monotonic (DM), Earliest Deadline First (EDF), and Least Laxity First (LLF). The RTOS kernel must allow for full hardware resource control and ability to override any built-in operating system resource management. It must fully support POSIX 1003.1b synchronous and asynchronous intertask communication, control, and scheduling (Applications, Committee, Computer, & Group, 2008)⁠. It must be able to lock memory address ranges into cache, lock memory address ranges in working memory if virtual, implement memory paging, and have high-precision time-stamping, interval timers, and real-time clocks and virtual timers.

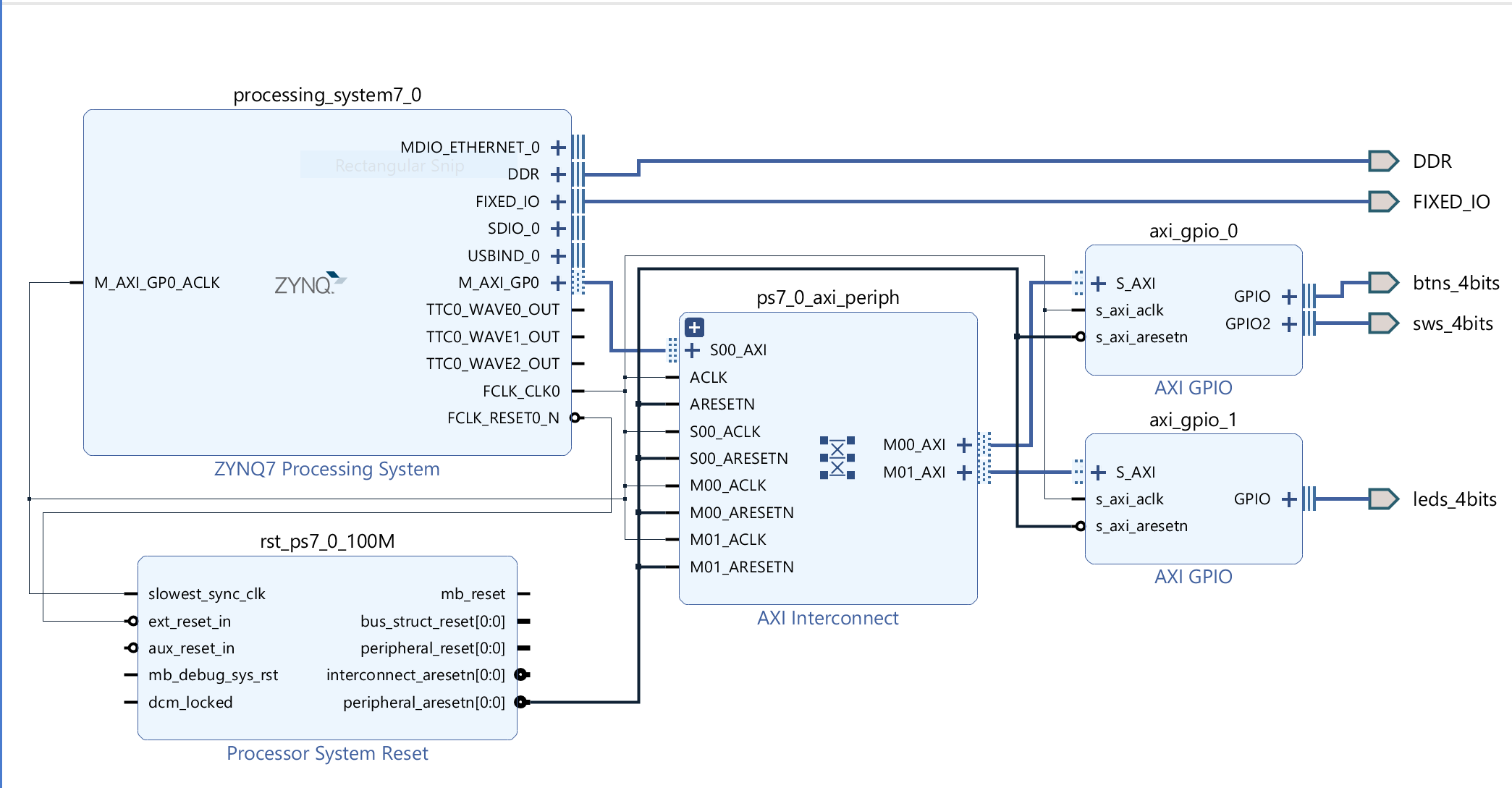
In general, an RTOS provides a threading mechanism, sometimes referred to as a task context which is the implementation of a service. A service is the execution context. In preemptive kernels, a context can be suspended through an asyncronous interrupt. The context state is put on a stack, the interrupted service goes until completion, unless preempted by a higher priority context. If it finishes, the last service added to the stack is recovered and run until completion or preemption. There are cases where this recovery could be non-recoverable, for example, division by zero. Because the very next insgtruction might cause a total system failure, a non-recoverable exception should result in thread suspension. An example of this state-flow is shown in figure 1.

  
Figure 1: Service States Including Programmed Suspension or Suspension Due to Exception

**Discussion**

The first step in this lab is to create a block design as seen below in figure 2.

Then, after bitstream creation, and SDK application using FreeRTOS and the hello world example code is selected. Running this on the board prints, “Hello from FreeRTOS example main,” then prints, “Rx task received string from Tx task: Hello from FreeRTOS example main.” If this happens 10 times in less than 10 seconds the string, “FreeRTOS Hello World Example PASSED,” and, “FreeRTOS Hello World Example FAILED,” if this is criteria is not met. This is controlled by the timer task TIMER\_CHECK\_THRESHOLD being compared to RxtaskCntr, which increments each time the RX task has received the string from the queue. Below will be a list of implemented functions built in to FreeRTOS:

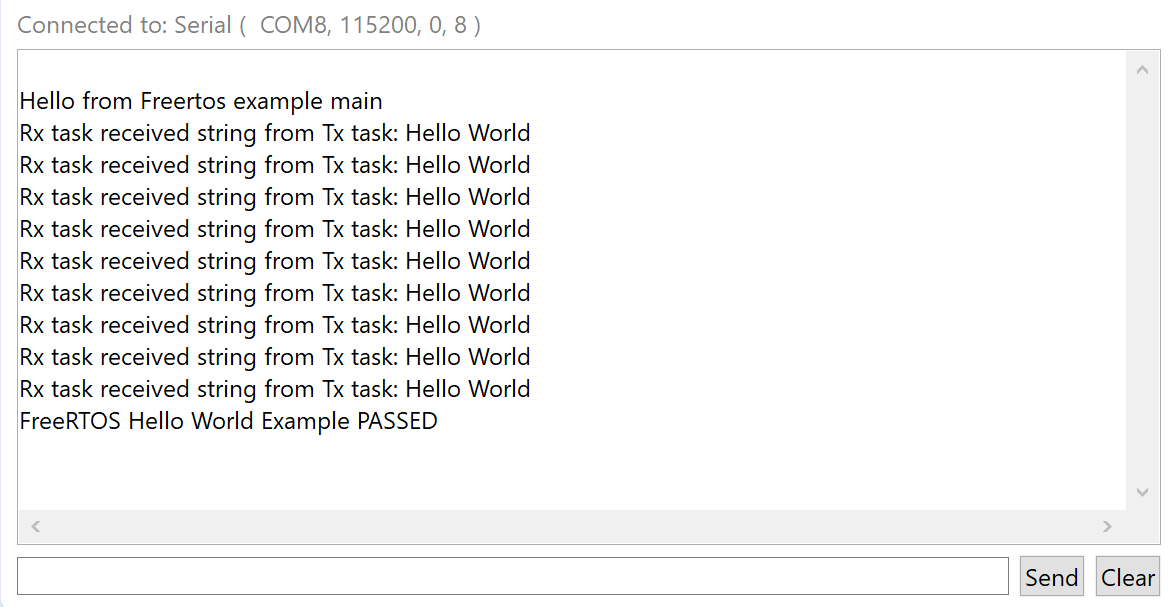
  
Figure 2: Block Design

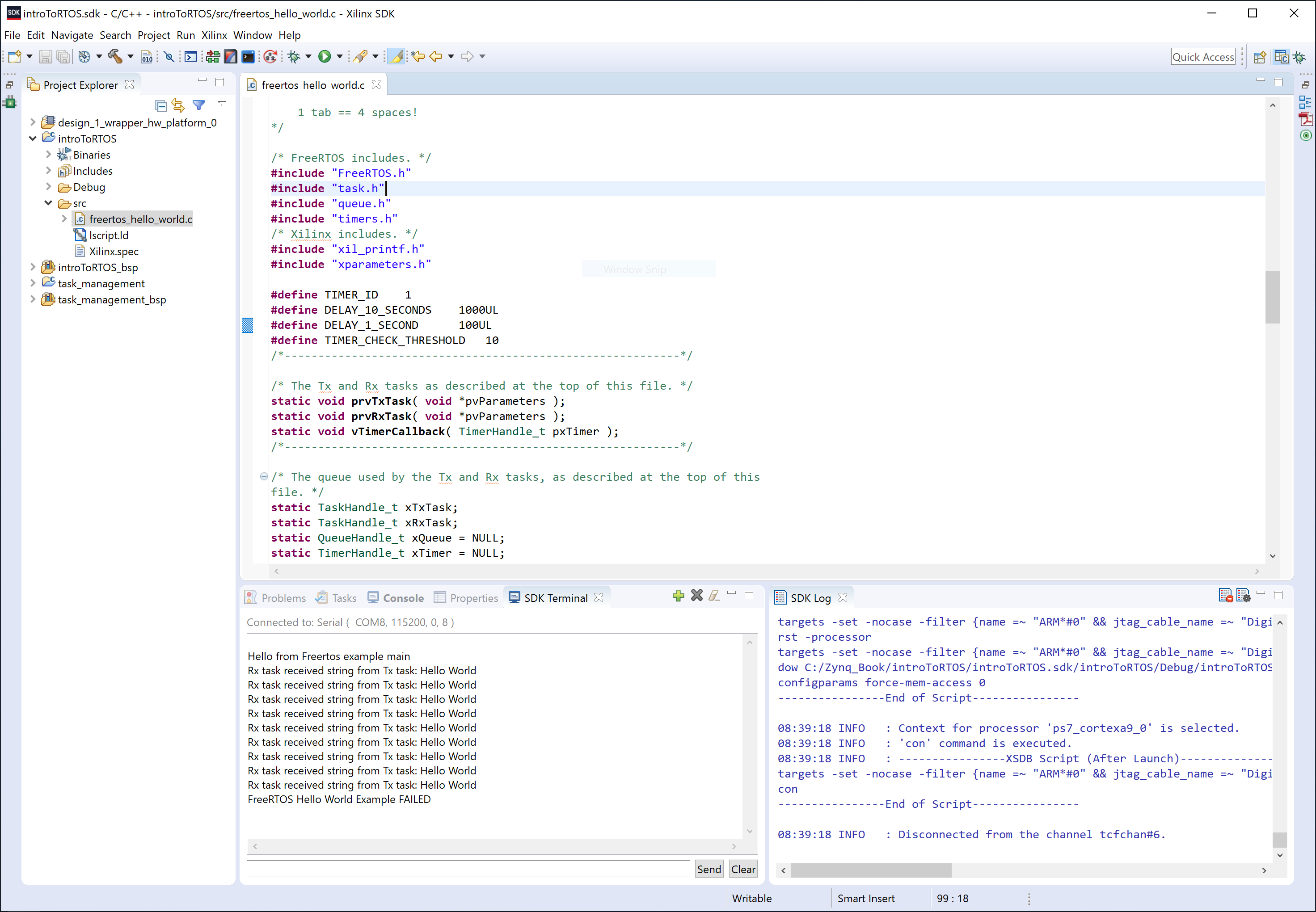
* xTaskCreate: If configSUPPORT\_DYNAMIC\_ALLOCATION == 1, this function is used to create a new task taking as arguments: the task handle, a debugging string, the stack allocation size, a (void\*) parameter, a priority level, and the static TaskHandle\_T object, which is the type by which tasks are referred.
* xQueueCreate: Takes in as arguments: the size of the queue, and size of the item(s) on the queue. This is actually a macro which calls xQueueGenericCreate, taking in the same arguments, but also taking in the ucQueueType, which is set internally.
* configASSERT: checks if the queue was actually created.
* xTimerCreate: Takes in as arguments: timer name, timer period in ticks, auto-reload setting, a timer id pointer, and the timer callback function.
* xTimerStart: this is a macro that takes the handle of the timer being started, and xTicksToWait which specifies the time, in ticks, that the calling task should \* be held in the Blocked state to wait for the start command to be successfully \* sent to the timer command queue, should the queue already be full when \* xTimerStart() was called. xTicksToWait is ignored if xTimerStart() is called \* before the scheduler is started. It returns the pass or fail.
* vTaskStartScheduler: as the name implies, it starts the scheduler.
* vTaskDelay: delays task for given amount of tics
* xQueueSend: Macro that sends the next value on the queue; takes as arguments: the queue being written to, the address of the data to be sent, and the maximum amount of time the task should block waiting for space to become available on the queue, should it already be full in number of tics.
* xQueueReceive: the mirror opposite of xQueueSend.
* vTaskDelete: takes the task handle to be deleted; if null, the calling task is deleted.
* pdMS\_TO\_TICKS: Converts a time in milliseconds to a time in ticks.

**Conclusions**

This lab is an introduction to FreeRTOS on the Zybo board. FreeRTOS extends the capabilities of the Zybo board whilst also making development easier. RTOS systems are used in many different industries, including highly specialized, safety-critical systems. An example of a hard-time deadline RTOS system is the one that controls automotive airbag systems. The time to inflation must meet a deadline; if it misses this, it can lead to injury or death.

**Appendices**

  
Figure 3: Hello World Pass

  
Figure 4: Hello World Fail

**Bibliography**

Applications, P., Committee, S., Computer, I., & Group, T. O. (2008). *IEEE Standard for Information Technology- Portable Operating System Interface (POSIX) Base Specifications, Issue 7*. *IEEE Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004)*. https://doi.org/10.1109/IEEESTD.2008.4694976

Liu, C. L., & Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, *20*(1), 46–61. https://doi.org/10.1145/321738.321743

Siewert Sam., P. J. (2016). Real-time embedded components and systems : with Linux and RTOS.