Lab 2: G8RTOS SCHEDULER AND SEMAPHORES

OBJECTIVES

In this lab, you will implement the structures and functionality necessary for the main component of your G8RTOS, the Thread Scheduler. You will also implement simple spin-lock semaphores to allow for thread synchronization and mutually exclusive access of resources

REQUIRED MATERIALS

Hardware

- MSP432 Launchpad
- Sensors Booster Pack
- LED array module

Software

- Board Support Package (BSP)
- G8RTOS_Empty
- Lab 1 LED library

Part 1: Setting up BSP

You will follow these steps whenever making a project using the BSP

1. Create a new CCS Project

2. **Adding BSP to Project:**
   Add the BoardSupportPackage folder (found in BSP.zip) to your project by dragging it into CCS. Choose to link the files over.
   Note: Be sure that your BoardSupportPackage folder is in a known file location. You will need to use this in the future!

3. **Updating your includes:**
   a. Right-click on your project and select *Properties*.

   b. Go to *Build->Include Options*.

   c. In the "Add dir" section, click on the green plus in the upper right hand corner.

   d. Locate and add the BoardSupportPackage/DriverLib folder

   e. Click on the green plus again to add another directory

   f. Locate and add the BoardSupportPackage/inc folder

4. At this point, you should be able to build your project with only 2 warnings from the library.
   [variable "array" was set but never used]
   [variable "v_pre_config_value_u8" was set but never used]
   You can ignore these.

Part 2: Integrating LED Library into BSP

The LED Drivers you created in your previous lab will need to be integrated into the BSP. To do this, you will need to accomplish these steps:

1. Add your C and H files to the BoardSupportPackage src and inc folders in your project.
2. In BSP.h add an #include to your driver's H file
3. In BSP.c add your initialization function into the BSP_InitBoard function

This is a good time to get familiar with the BSP by writing some small demo applications.

At this point you are ready to begin designing your Operating System. Before starting, you will need to add the G8RTOS_Empty folder into your current project. You will also need to add the G8RTOS_Empty folder to your includes.

Part 3: Create OS structures

File: G8RTOS_Structures.h

The Thread Control Block (TCB) is responsible for holding all relevant information regarding the status of a given thread. For this lab, it should contain the following fields:

a. Next TCB pointer
b. Previous TCB pointer
c. Stack pointer for the thread's stack

You will add additional fields in subsequent labs.

Create the TCB structure in the appropriate header file. See comments for supporting details.

Part 4: Fill in the two Initialization functions

File: G8RTOS_Scheduler.c

Lab 2: G8RTOS SCHEDULER AND SEMAPHORES

1. InitSysTick
    a. Initialize SysTick to overflow every 1ms
    b. You may use **ClockSys_GetSysFreq()** to get the current clock speed in Hz
2. G8RTOS_Init
   Before configuring your operating system, you must initialize it to a known starting state. The G8RTOS_Init function will be the first function called in your project to accomplish this. The function needs to accomplish the following:
    a. Initialize system time to zero
    b. Set the number of threads to zero
    c. Initialize all hardware on the board

Part 5: Implement G8RTOS_AddThread
File: G8RTOS_Scheduler.c

Before launching your operating system, you must add threads to the scheduler. The G8RTOS_AddThread function will take in a void/void function pointer to insert the thread into the scheduler. It should accomplish:

a. Initializing a TCB for the given thread
b. Adding the TCB to the Round-Robin scheduler list
c. Initializing the thread stack to hold a default thread register context

Part 6: Implement Exception Handlers
Files: G8RTOS_Scheduler.c and G8RTOS_SchedulerASM.s

To accomplish multithreading, you must use exceptions begin the context switches.

The PendSV_Handler will be used to save a thread's context, call the scheduler, and load the next scheduled thread's context.

You will need to write this handler in assembly, in G8RTOS_SchedulerASM.s, to have direct access to the CPU's registers.

The SysTick_Handler in G8RTOS_Scheduler.c will be used to provide a constant quantum for each thread before preemption. For this lab, this handler will simply trigger the PendSV exception. In the future, this handler will have more responsibilities.

Part 7: Implement G8RTOS_Scheduler
File: G8RTOS_Scheduler.c

The scheduler will be called by the PendSV_Handler and will be responsible for choosing the next TCB to run. For this lab's Round-Robin scheduler, the G8RTOS_Scheduler function will set the pointer, CurrentlyRunningThread, to the currently running thread's 'next' (nextTCB).

Part 8: Implement G8RTOS_Launch and G8RTOS_Start
Files: G8RTOS_Scheduler.c and G8RTOS_SchedulerASM.s

When you have finished configuring G8RTOS, your OS is now ready to launch. To start the OS, you must arm the SysTick and PendSV exceptions, set the CurrentlyRunningThread to the first thread in the scheduler, load the context of said thread into the CPU, and enable interrupts. This task will be split into two functions.

1. G8RTOS_Launch:
   This C function will be called from main and will accomplish the following:
    a. Set CurrentlyRunningThread
    b. Initialize SysTick
    c. Set the priorities of PendSV and SysTick to the lowest priority
    d. Call G8RTOS_Start
2. G8RTOS_Start:
   This assembly function will accomplish the following:
    a. Loads the currently running thread's context into the CPU
    b. Enable interrupts

Part 9: Testing the scheduler
At this point you will write three simple threads to make sure your scheduler works before adding any semaphores or peripherals.

Main creation:

1. Create a main.c file in your project's root directory. The main needs to:
    a. Create three functions called task0, task1, and task2
    b. Each task function will increment their own counter variable. Name them accordingly (i.e. task0 will

Lab 2: G8RTOS SCHEDULER AND SEMAPHORES

have counter0, task1 will have counter1, etc.)
   c. Call G8RTOS_Init
   d. Add the threads to your scheduler
   e. Launch the OS

Part 10: Implement semaphore functions
File: G8RTOS_Semaphores.c

You will now add the ability to synchronize threads and control peripheral access through a naive spinlock semaphore implementation.

1. G8RTOS_InitSemaphore
   This function will assign the semaphore pointer parameter the value of the value parameter. **Note: This should all be accomplished in a critical section!**

2. G8RTOS_WaitSemaphore
   This function will check if the given semaphore parameter is greater than 0. If it is not, it will constantly check until it is. During this "spinlock", **the function will exit and reenter a critical section** to allow other threads to run. After the spinlock, the semaphore will be decremented to indicate ownership of the semaphore.
3. G8RTOS_SignalSemaphore
   This function will increment the semaphore to indicate releasing ownership of the semaphore. **Note: This is also a critical section!**

Part 11: Create a main file and three threads
Thread descriptions:

1. Thread 0:
   a. Wait for the sensor I2C semaphore.
   b. Read from the accelerometer's x-axis and save the value into a local variable
   c. Release the sensor I2C semaphore
   d. Wait for the LED I2C semaphore
   e. Output data to Red LEDS as shown in Figure A.
   f. Release the LED I2C semaphore

2. Thread 1:
   a. Wait for the sensor I2C semaphore
   b. Read from the light sensor and save value into a local variable
   c. Release the sensor I2C semaphore
   d. Wait for the LED I2C semaphore
   e. Output data to Green LEDS as shown in Figure B.
   f. Release the LED I2C semaphore
3. Thread 2:
   a. Wait for the sensor I2C semaphore
   b. Read from the gyro's z-axis and save value into a local variable
   c. Wait for LED I2C semaphore
   d. Output data to Blue LEDS as shown in Figure C.
   e. Release LED I2C semaphore
4. Threads.h will hold extern declarations of all the threads as well as extern declarations of the sensor and LED semaphores.

Thread creation:

1. Create a Threads.c to hold all threads and semaphores in the root directory
2. Create a Threads.h file that references all the semaphores to be initialized and references the threads for main to see

Main:

2. Update your main.c file in your project's root directory to do the following:
   a. Call G8RTOS_Init
   b. Initialize the semaphores in Threads.h
   c. Add the threads in Threads.h to the scheduler
   d. Launch the OS

Project Explorer
At this point, your project explorer should resemble the following:

Project:
- BSP Folder
- G8RTOS Folder
- main.c
- Threads.h
- Threads.c

Lab 2: G8RTOS SCHEDULER AND SEMAPHORES

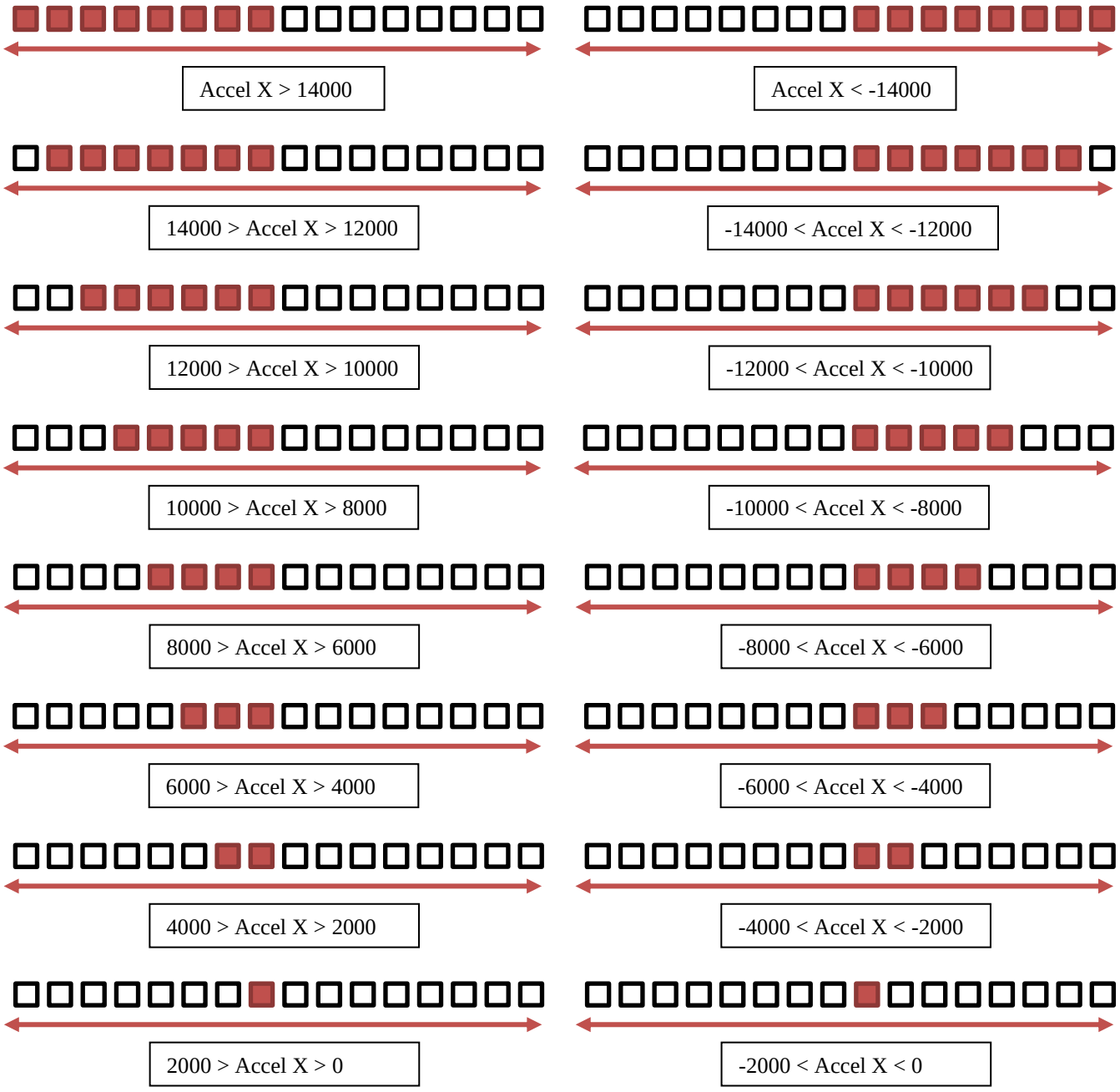

Figure A.

Lab 2: G8RTOS SCHEDULER AND SEMAPHORES

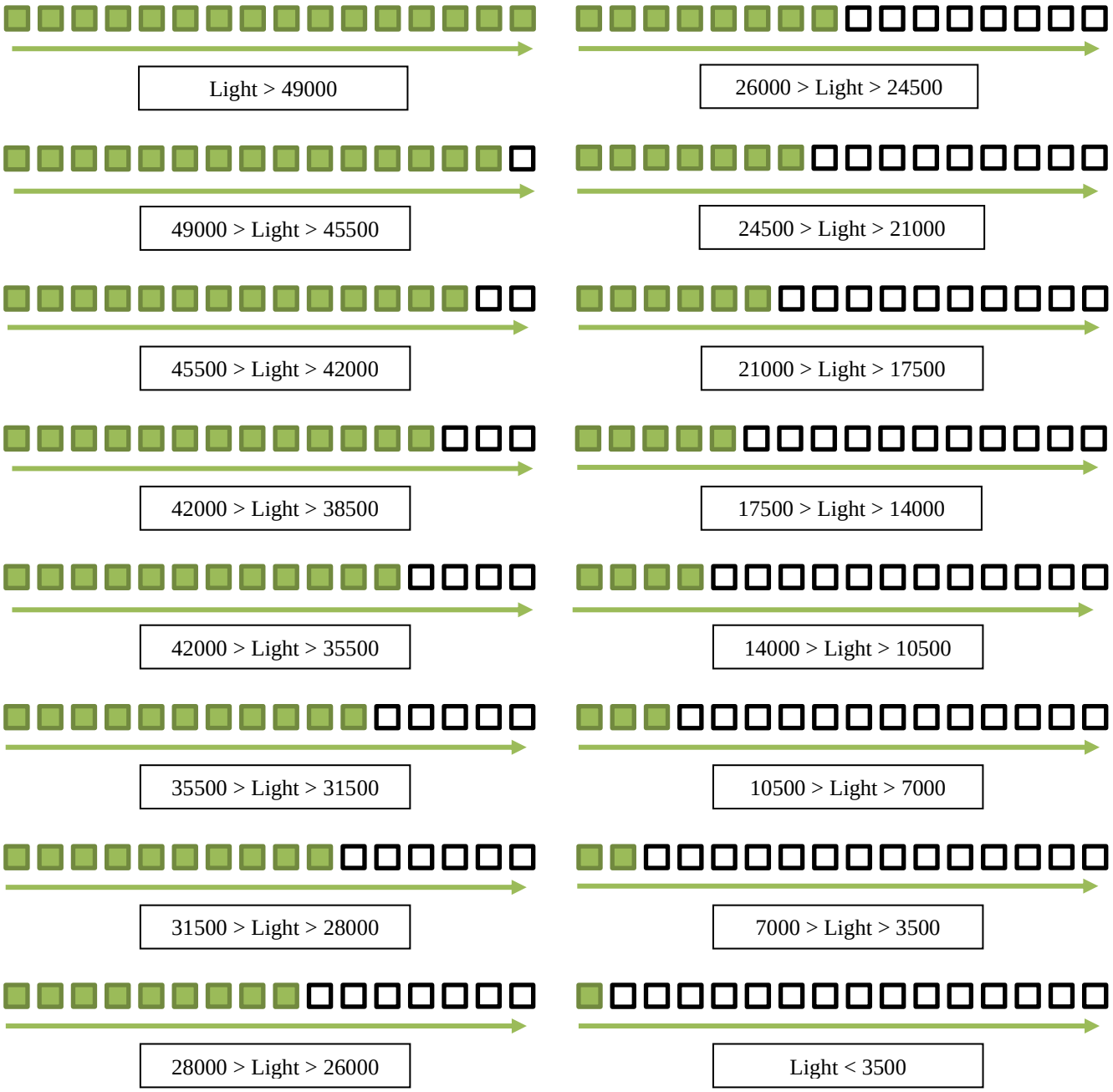

Figure B.

Lab 2: G8RTOS SCHEDULER AND SEMAPHORES

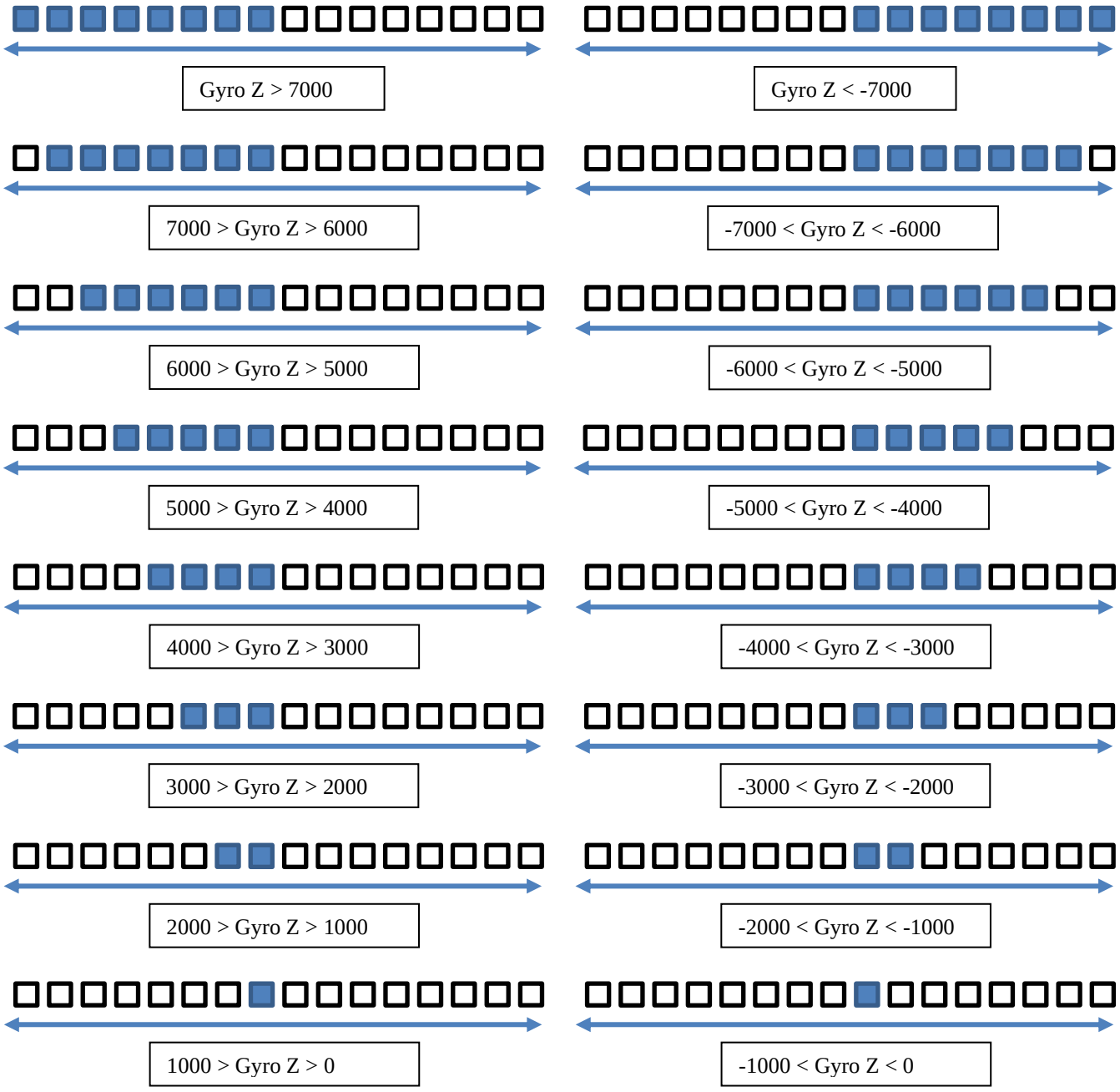

Figure C.