

CPE403/ECG603/ECG710 – Advanced

Embedded Systems/Real-Time Embedded Systems

Design Assignment 3

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Joshua Martinez

Email: marti87@unlv.nevada.edu

Github Repository link (root): https://github.com/JoshuaMa2003/submission_da_0011

Youtube Playlist link (root):

https://www.youtube.com/playlist?list=PLZ09MA_uFt61Vw2JCAtw-8vAdmoCKXpZ

Follow the submission guideline to be awarded points for this Assignment.

Submit the following for all Assignments:

1. In the document, for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include comments. If no base code is provided, submit the base code for the first task only.
2. Create a private Github repository with a random name (no CPE403/603/710, Lastname, Firstname). Place all labs under the root folder MSP432E4/CC1352, sub-folder named Assignment1, with one document and one video link file for each lab, place modified c files named as asng_taskxx.c.
3. If multiple c files or other libraries are used, create a folder asng1_t01 and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup_ccs.c and other include files, c) text file with youtube video links (see template).
5. Submit the doc file in canvas before the due date. The root folder of the github assignment directory should have the documentation and the text file with youtube video links.
6. Organize your videos as a playlist under the name “ADVEMBSYS”. The playlist should

have the video sequence arranged as submission or due dates.

7. Only submit the PDF. Upload this document in the github repository and in canvas.

1. Code for tasks: For each task, submit the relevant modified/section or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the initialization and execution section of each task separately. Use a separate page for each task.

Task 1 – ADC Task (5ms, ADC0 CH0 on joystick)

- **Original Code:**

The TI-RTOS mutex_MSP_ESP432E401Y_tirtos_ccs example did not include any ADC usage. It only had two generic tasks (taskFxn0, taskFxn1) that printed messages and used a mutex.

- **Modified Code:**

```
45 /*
46 * ===== Configuration constants =====
47 */
48
49 #define TASKSTACKSIZE          2048    // Stack size for each task
50 #define SYSTEM_TICK_MS         1        // BIOS Clock tick (1 ms)
51
52 #define ADC_PERIOD_MS          5        // ADC task release period
53 #define UART_PERIOD_MS         10       // UART task release period
54 #define PWM_PERIOD_MS          15       // PWM task release period
55 #define HEARTBEAT_PERIOD_MS    500      // Heartbeat blink period
56
57 #define ADC_MAX_VALUE          4095U   // 12-bit ADC full-scale value
58
59 /*
60 * ===== Globals: shared data and handles =====
61 */
62
63 /* Most recent ADC sample (updated by ADC task, read by others) */
64 static volatile uint16_t gAdcValue = 0;
65
66 /* ADC value latched by the switch HWI for use by the PWM task */
67 static volatile uint16_t gLatchedAdcValue = 0;
68
69 /* Flag set in Hwi when a new PWM update is pending */
70 static volatile bool gPwmUpdatePending = false;
71
72 /* Driver handles (initialized in main) */
73 static UART_Handle uartHandle = NULL;
74 static PWM_Handle  pwmHandle  = NULL;
75
76 /* Semaphore handles (obtained from the constructed structs) */
77 static Semaphore_Handle adcSem      = NULL;
78 static Semaphore_Handle uartSem     = NULL;
79 static Semaphore_Handle pwmSem     = NULL;
80 static Semaphore_Handle adcDataMutex = NULL;
81 static Semaphore_Handle uartMutex   = NULL;
82
```

```

131/*
132 * ===== Clock callbacks (1 ms base tick → 5/10/15/500 ms periods) =====
133 */
134
135 static void adcClockFxn(UArg arg0)
136{
137    (void)arg0;
138    /* Trigger ADC Task at 5 ms instances of the 1 ms system tick */
139    if (adcSem != NULL) {
140        Semaphore_post(adcSem);
141    }
142}
143

148/*
149 * ===== ADC Task =====
150 *
151 * - Waits on adcSem.
152 * - Every 5 ms (Clock driven), performs an ADC conversion on ADC0-CH0.
153 * - Stores the result in gAdcValue under protection of adcDataMutex.
154 */
155

156 static void adcTaskFxn(UArg arg0, UArg arg1)
157{
158    (void)arg0;
159    (void)arg1;
160
161    ADC_Handle adcHandle;
162    uint16_t sample;
163
164    /* Open ADC driver instance configured as CONFIG_ADC_0 in SysConfig.
165     * Make sure CONFIG_ADC_0 is mapped to PE3/AIN0 in your .syscfg file.
166     */
167    adcHandle = ADC_open(CONFIG_ADC_0, NULL);
168    if (adcHandle == NULL) {
169        /* If ADC fails to open, halt here */
170        while (1) { }
171    }
172
173    for (;;) {
174        /* Block until the 5 ms Clock posts the semaphore */
175        Semaphore_pend(adcSem, BIOS_WAIT_FOREVER);
176
177        /* Single ADC conversion */
178        if (ADC_convert(adcHandle, &sample) == ADC_STATUS_SUCCESS) {
179
180            /* Protect the shared variable while updating it */
181            Semaphore_pend(adcDataMutex, BIOS_WAIT_FOREVER);
182            gAdcValue = sample;
183            Semaphore_post(adcDataMutex);
184
185        }
186    }
187}
188
189

```

Description: The original mutex example had no ADC, so a new ADC task was created. A Clock object (*adcClockStruct*) with a 5 ms period posts a counting semaphore (*adcSem*) that wakes the ADC task at every 5 ms tick. Inside *adcTaskFxn* I open *CONFIG_ADC_0* (configured in SysConfig as ADC0 CH0 / joystick input), perform *ADC_convert()*, and write the result to the shared global *gAdcValue*. A binary semaphore (*adcDataMutex*) is used as a mutex so that other tasks (UART, PWM) can safely read this shared ADC value without race conditions.

Task 2 – UART Display Task (10 ms)

- Original Code:

The mutex example only contained simple UART prints inside its demo tasks. These were not synchronized to a specific period and did not share ADC data.

- Modified Code:

```
45 /*
46 * ===== Configuration constants =====
47 */
48
49 #define TASKSTACKSIZE      2048    // Stack size for each task
50 #define SYSTEM_TICK_MS     1        // BIOS Clock tick (1 ms)
51
52 #define ADC_PERIOD_MS      5        // ADC task release period
53 #define UART_PERIOD_MS     10       // UART task release period
54 #define PWM_PERIOD_MS      15       // PWM task release period
55 #define HEARTBEAT_PERIOD_MS 500     // Heartbeat blink period
56
57 #define ADC_MAX_VALUE      4095U   // 12-bit ADC full-scale value
58
59 /**
60 * ===== Globals: shared data and handles =====
61 */
62
63 /* Most recent ADC sample (updated by ADC task, read by others) */
64 static volatile uint16_t gAdcValue = 0;
65
66 /* ADC value latched by the switch HWI for use by the PWM task */
67 static volatile uint16_t gLatchedAdcValue = 0;
68
69 /* Flag set in Hwi when a new PWM update is pending */
70 static volatile bool gPwmUpdatePending = false;
71
72 /* Driver handles (initialized in main) */
73 static UART_Handle uartHandle = NULL;
74 static PWM_Handle  pwmHandle  = NULL;
75
76 /* Semaphore handles (obtained from the constructed structs) */
77 static Semaphore_Handle adcSem      = NULL;
78 static Semaphore_Handle uartSem    = NULL;
79 static Semaphore_Handle pwmSem    = NULL;
80 static Semaphore_Handle adcDataMutex = NULL;
81 static Semaphore_Handle uartMutex  = NULL;
82
83
84 /**
85 * Function: uartClockFxn
86 * Description: Trigger UART Task at 10 ms instances
87 */
88 static void uartClockFxn(UArg arg0)
89 {
90     (void)arg0;
91     /* Trigger UART Task at 10 ms instances */
92     if (uartSem != NULL) {
93         Semaphore_post(uartSem);
94     }
95 }
```

```

229 /*
230 * ====== UART Task ======
231 *
232 * - Waits on uartSem.
233 * - Every 10 ms, reads current ADC value from gAdcValue
234 *   (guarded by adcDataMutex).
235 * - Prints the value and percentage over UART.
236 * - Uses uartMutex so PWM task and UART task do not interleave text.
237 *
238 */
239
240 static void uartTaskFxn(UArg arg0, UArg arg1)
241{
242     (void)arg0;
243     (void)arg1;
244
245     uint16_t adcCopy;
246     char msg[64];
247
248     for (;;) {
249         /* Wait for 10 ms period tick */
250         Semaphore_pend(uartSem, BIOS_WAIT_FOREVER);
251
252         /* Take a safe snapshot of the ADC value */
253         Semaphore_pend(adcDataMutex, BIOS_WAIT_FOREVER);
254         adcCopy = gAdcValue;
255         Semaphore_post(adcDataMutex);
256
257         /* Convert reading to % of full-scale for easier interpretation */
258         uint32_t adcPercent =
259             ((uint32_t)adcCopy * 100U) / ADC_MAX_VALUE;
260
261         int len = sprintf(msg, sizeof(msg),
262                           "[UART] ADC = %4u (%2lu%%)\r\n",
263                           adcCopy,
264                           (unsigned long)adcPercent);
265
266         if (len > 0) {
267             /* UART is a shared resource - guard with mutex semaphore */
268             Semaphore_pend(uartMutex, BIOS_WAIT_FOREVER);
269             UART_write(uartHandle, msg, (size_t)len);
270             Semaphore_post(uartMutex);
271         }
272     }
273 }
274

```

Description: Instead of simple prints from the original demo tasks, I created a dedicated UART task that is triggered every 10 ms via a Clock + semaphore. The task acquires *adcDataMutex*, reads a snapshot of *gAdcValue*, converts it into a percentage of ADC full-scale, and formats a status line. All UART writes are serialized through *uartMutex*, so this task and the PWM task can both print without interleaving characters.

Task 3 – PWM Task (PFO, 15 ms)

- **Original Code:**

The mutex template did not generate PWM; PFO was unused. So PWM setup and PFO behavior were added entirely.

- **Modified Code:**

```
45 /*
46 * ===== Configuration constants =====
47 */
48
49 #define TASKSTACKSIZE          2048    // Stack size for each task
50 #define SYSTEM_TICK_MS         1        // BIOS Clock tick (1 ms)
51
52 #define ADC_PERIOD_MS          5        // ADC task release period
53 #define UART_PERIOD_MS         10       // UART task release period
54 #define PWM_PERIOD_MS          15       // PWM task release period
55 #define HEARTBEAT_PERIOD_MS    500      // Heartbeat blink period
56
57 #define ADC_MAX_VALUE          4095U   // 12-bit ADC full-scale value
58
59 /**
60 * ===== Globals: shared data and handles =====
61 */
62
63 /* Most recent ADC sample (updated by ADC task, read by others) */
64 static volatile uint16_t gAdcValue = 0;
65
66 /* ADC value latched by the switch HWI for use by the PWM task */
67 static volatile uint16_t glatchedAdcValue = 0;
68
69 /* Flag set in Hwi when a new PWM update is pending */
70 static volatile bool gPwmUpdatePending = false;
71
72 /* Driver handles (initialized in main) */
73 static UART_HandleTypeDef uartHandle = NULL;
74 static PWM_HandleTypeDef pwmHandle = NULL;
75
76 /* Semaphore handles (obtained from the constructed structs) */
77 static Semaphore_HandleTypeDef adcSem      = NULL;
78 static Semaphore_HandleTypeDef uartSem     = NULL;
79 static Semaphore_HandleTypeDef pwmSem     = NULL;
80 static Semaphore_HandleTypeDef adcDataMutex = NULL;
81 static Semaphore_HandleTypeDef uartMutex   = NULL;
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154 static void pwmClockFxn(UArg arg0)
155 {
156     (void)arg0;
157     /* Trigger PWM Task at 15 ms instances */
158     if (pwmSem != NULL) {
159         Semaphore_post(pwmSem);
160     }
161 }
```

```

275 /*
276 * ===== PWM Task =====
277 *
278 * - Waits on pwmSem (15 ms period).
279 * - Checks if the switch Hwi has latched a new ADC value.
280 * - If so, converts that latched ADC value to a PWM duty fraction
281 * and updates PWM on PF0 (CONFIG_PWM_0).
282 * - Also prints info to UART about the new duty cycle.
283 *
284 * interaction: PWM duty only changes when the switch has triggered
285 * and latched a value, and stays fixed otherwise.
286 */
287
288 static void pwmTaskFxn(UArg arg0, UArg arg1)
289 {
290     (void)arg0;
291     (void)arg1;
292
293     uint32_t currentDuty = 0;
294
295     /* Start with duty = 0% (LED off), per assignment */
296     PWM_setDuty(pwmHandle, currentDuty);
297
298     for (;;) {
299         /* Release every 15 ms */
300         Semaphore_pend(pwmSem, BIOS_WAIT_FOREVER);
301
302         /*
303         * Safely snapshot the "update pending" flag and the latched ADC
304         * value. We briefly disable interrupts to avoid racing the Hwi.
305         */
306         UInt key      = Hwi_disable();
307         bool pending  = gPwmUpdatePending;
308         uint16_t latchedVal = glatchedAdcValue;
309         if (pending) {
310             gPwmUpdatePending = false; // consume the event
311         }
312         Hwi_restore(key);
313
314         if (pending) {
315             /* Convert 12-bit ADC value into PWM fraction.
316             * PWM_DUTY_FRACTION_MAX is defined by the TI PWM driver when
317             * dutyUnits == PWM_DUTY_FRACTION.
318             */
319             uint64_t scaled =
320                 ((uint64_t)latchedVal * (uint64_t)PWM_DUTY_FRACTION_MAX) /
321                 (uint64_t)ADC_MAX_VALUE;
322
323             currentDuty = (uint32_t)scaled;
324             PWM_setDuty(pwmHandle, currentDuty);
325

```

```

26     /* Compute some human-readable info for UART */
27     char msg[96];
28     uint32_t dutyPercent =
29         ((uint32_t)((uint64_t)currentDuty * 100U) /
30          (uint64_t)PWM_DUTY_FRACTION_MAX);
31
32     Semaphore_pend(uartMutex, BIOS_WAIT_FOREVER);
33
34     int len = sprintf(msg, sizeof(msg),
35                         "[SW1] Latched ADC = %4u (%2lu%%)\r\n",
36                         latchedVal,
37                         (unsigned long)((uint32_t)latchedVal *
38                                         100U / ADC_MAX_VALUE));
39     if (len > 0) {
40         UART_write(uartHandle, msg, (size_t)len);
41     }
42
43     len = sprintf(msg, sizeof(msg),
44                     "[PWM] Duty = %4lu (%2lu%%)\r\n",
45                     (unsigned long)currentDuty,
46                     (unsigned long)dutyPercent);
47     if (len > 0) {
48         UART_write(uartHandle, msg, (size_t)len);
49     }
50
51     Semaphore_post(uartMutex);
52 }
53 }
54 }
55 }
```

Description: The PWM task is woken every 15ms by a Clock + semaphore, but it only changes PF0's duty cycle when the switch HWI has latched a new ADC value. The ISR stores the joystick reading into *gLatchedAdcValue* and sets *gPwmUpdatePending*. The PWM task enters a small critical section using *Hwi_disable()* / *Hwi_restore()* to automatically copy and clear the flag, then maps the 12-bit ADC reading to *PWM_DUTY_FRACTION_MAX* and calls *PWM_setDuty()*. This guarantees that the duty cycle changes only on switch events, and stays fixed between presses.

Task 4 – Switch Read Task (HWI on SW1)

- **Original Code (Template):**
The mutex template already had a GPIO button callback, but it only toggled an LED and posted a semaphore. For this, I changed the handler so that it captures the current ADC value for the PWM task.
- **Modified Code:**

```
179
180 void SwitchHwiFxn(uint_least8_t index)
181{
182    (void)index;
183
184    /* Latch the current ADC reading for the PWM task */
185    gLatchedAdcValue = gAdcValue;
186    gPwmUpdatePending = true; // mark that a new duty update is pending
187}

/*
 * ----- Hook SW1 to our Hwi callback and enable its interrupt -----
 */
99    GPIO_setCallback(CONFIG_GPIO_SW1, SwitchHwiFxn);
00    GPIO_enableInt(CONFIG_GPIO_SW1);
01
```

Description: Instead of toggling LEDs, the SW1 interrupt now serves as a Switch Read Task in hardware-interrupt form. When the user presses the switch, *SwitchHwiFxn* runs immediately, grabs the latest ADC sample (*gAdcValue*), and stores it in *gLatchedAdcValue* while also setting *gPwmUpdatePending = true*. The PWM task checks this flag on its 15 ms wakeup and uses that latched value to compare a new duty cycle. Because the ADC task runs periodically and the HWI only latches on button events, the PWM output remains fixed until the next switch press even though the ADC continues sampling.

Task 5 – Heartbeat Function (PF4 LED)

- **Original Code (Template):**

The mutex template used the board LED as a simple “alive” indicator in the tasks. For this, I repealed that with a separate Clock-driven heartbeat on PF4 so the LED is independent of the other tasks’s timing.

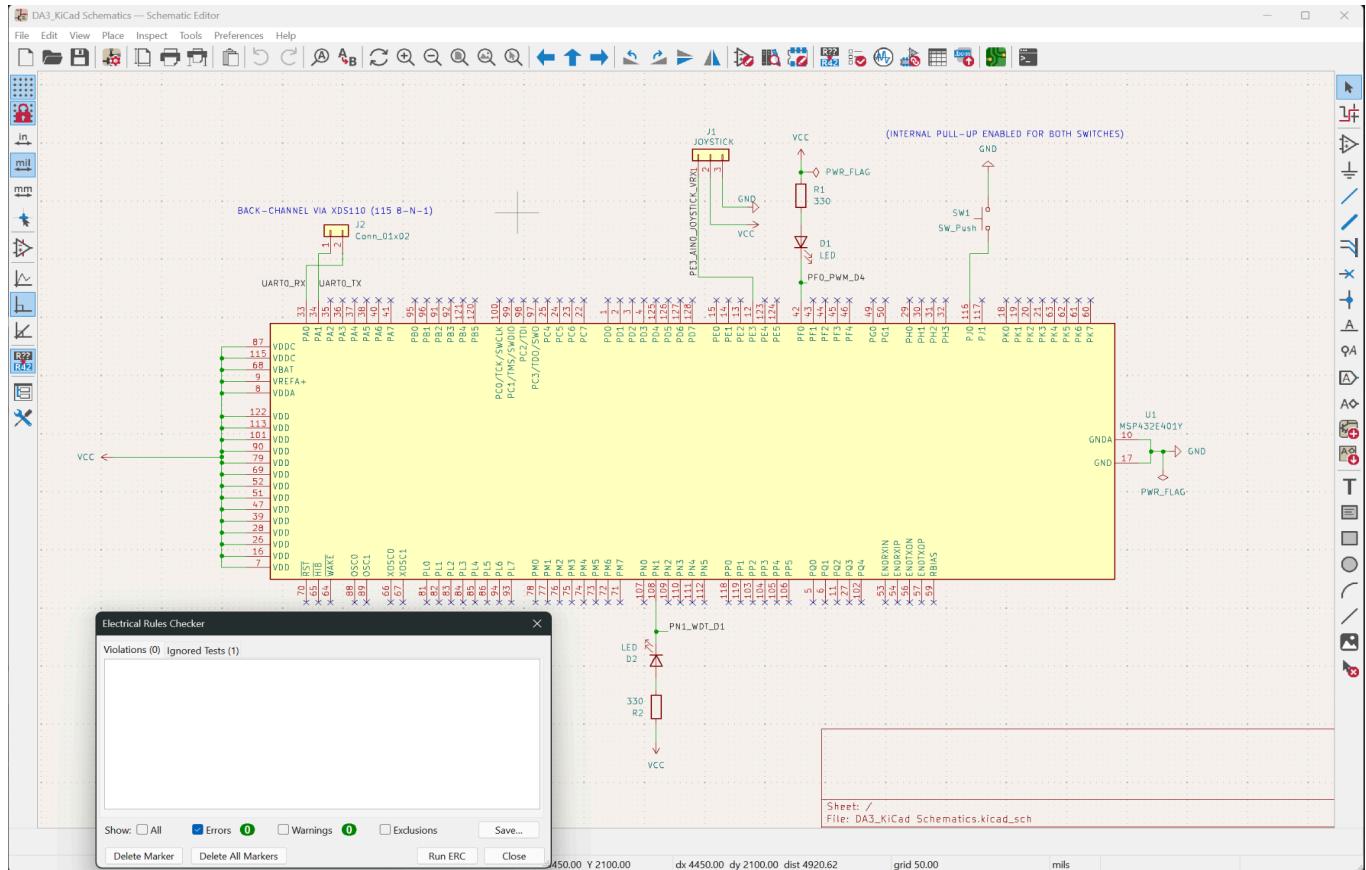
- **Modified Code:**

```
45 /*  
46 * ===== Configuration constants =====  
47 */  
48  
49 #define TASKSTACKSIZE      2048    // Stack size for each task  
50 #define SYSTEM_TICK_MS     1        // BIOS Clock tick (1 ms)  
51  
52 #define ADC_PERIOD_MS      5        // ADC task release period  
53 #define UART_PERIOD_MS     10       // UART task release period  
54 #define PWM_PERIOD_MS      15        // PWM task release period  
55 #define HEARTBEAT_PERIOD_MS 500      // Heartbeat blink period  
56  
57 #define ADC_MAX_VALUE      4095U   // 12-bit ADC full-scale value  
58  
59  
163 static void heartbeatClockFxn(UArg arg0)  
164{  
165     (void)arg0;  
166     /* Heartbeat: blink PF4 LED independently of other tasks */  
167     GPIO_toggle(CONFIG_GPIO_LED_0);  
168}  
169  
170  
171  
172  
173  
174     /* Heartbeat Clock: 500 ms period */  
175     Clock_Params hbClkParams;  
176     Clock_Params_init(&hbClkParams);  
177     hbClkParams.period    = HEARTBEAT_PERIOD_MS / SYSTEM_TICK_MS;  
178     hbClkParams.startFlag = TRUE;  
179     Clock_construct(&heartbeatClockStruct,  
180                     heartbeatClockFxn,  
181                     HEARTBEAT_PERIOD_MS / SYSTEM_TICK_MS,  
182                     &hbClkParams);  
183
```

Description: In order to meet the “heartbeat” requirement, I created a dedicated Clock object that fires every 500 ms, independent of the ADC/UART/PWM scheduling. Its callback *heartbeatClockFxn* simply toggles *CONFIG_GPIO_LED_0* (PF4), producing a steady blink that shows the RTOS is running even when other tasks are blocked or busy. Because it is Clock-driven, this heartbeat does not interfere with the timing of ADC acquisitions, UART prints, or PWM updates.

2. Block diagram and/or Schematics showing the components, pins used, and interface. You can use KiCAD/Eagle/Altium to get the schematics. KiCAD Symbol libraries for TI uCs are @ https://kicad.github.io/symbols/MCU_Texas.html and <https://www.snapeda.com/part/CC1352P1F3RGZT/Texas%20Instruments/view-part/>

KiCad Design Schematics (ERC Compliant):

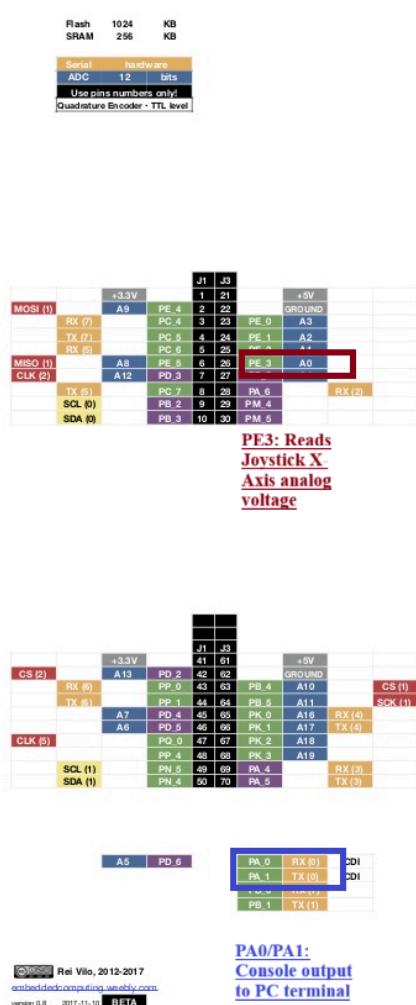


Block Diagram:



LaunchPad with MSP432E401Y

Revision 1



3. Screenshots of the IDE, physical setup, debugging process – Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.

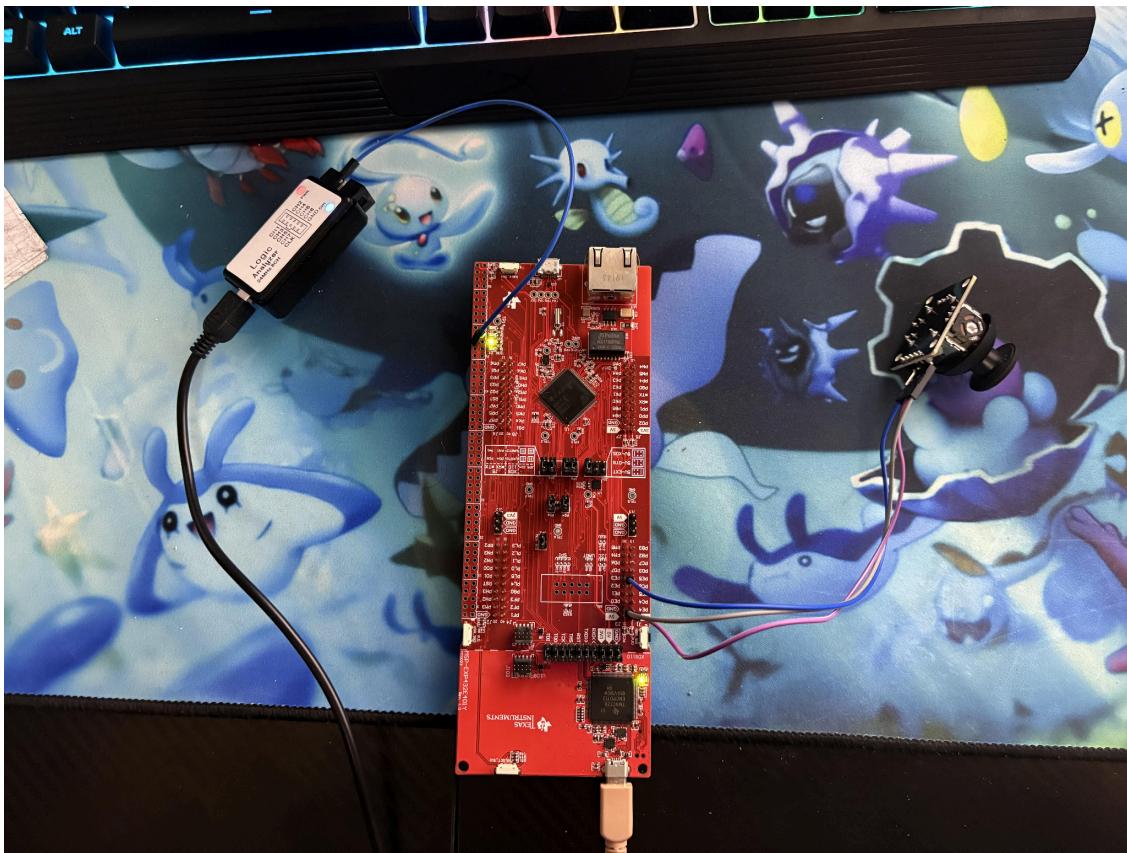
IDE Workspace:

The screenshot shows the Eclipse IDE interface with the CDT (C/C++ Development Tools) plugin. The Project Explorer view on the left lists the project structure for 'CPE403_DA3 [Active - Debug]'. The main editor window displays the 'Idle.c' source code, which includes comments and code for an idle loop. The Console view at the bottom shows the output of a build command, indicating a successful build of the configuration 'Debug' for the project 'CPE403_DA3'. The build command used was 'C:\ti\ccs1281\ccs\utils\bin\gmake' with options '-k -j 24 all -O'.

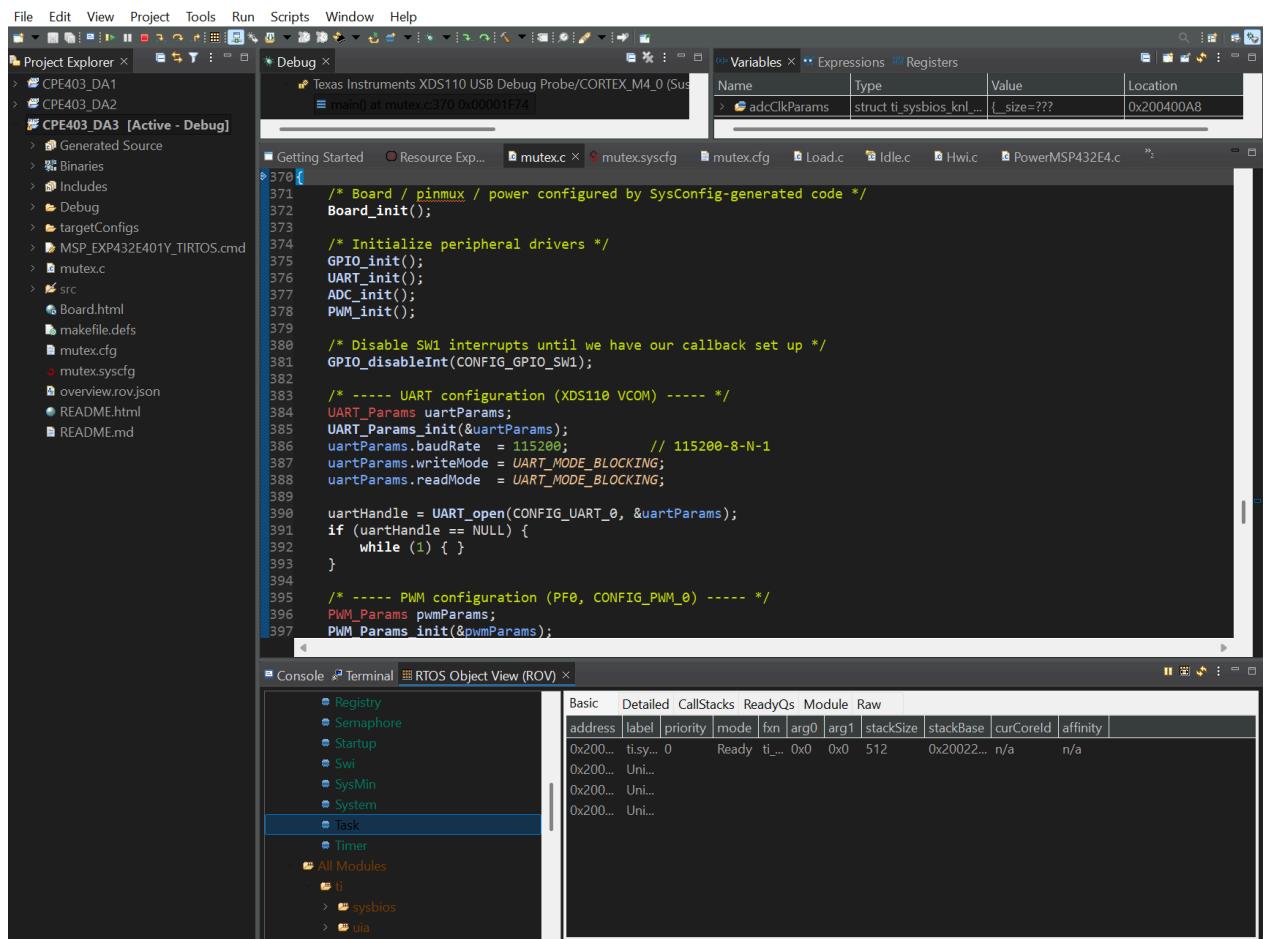
```
Getting Started Resource Exp... mutex.c mutex.syscfg mutex.cfg Load.c Idle.c Hwi.c
27 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
28 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
29 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
30 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
31 */
32 /*
33 * ===== Idle.c =====
34 * Implementation of Idle_loop.
35 */
36
37 #include <xdc/std.h>
38
39 #include <ti/sysbios/BIOS.h>
40
41 #include <ti/sysbios/hal/Core.h>
42 #include <ti/sysbios/hal/Hwi.h>
43
44 #include "package/internal/Idle.xdc.h"
45
46 /*
47 * ===== Idle_loop =====
48 */
49 /* MISRA.FUNC.UNUSEDPAR.2012 */
50 Void Idle_loop(UArg arg1, UArg arg2)
51{
52     /* INFINITE_LOOP.LOCAL */
53     while (TRUE) {
54         Idle_run();
55     }
56 }
57
58 */
59
***** Build of configuration Debug for project CPE403_DA3 *****
"C:\ti\ccs1281\ccs\utils\bin\gmake" -k -j 24 all -O
making ..\src\sysbios\sysbios.aem4f ..
gmake[1]: Nothing to be done for 'all'.
making ..\src\sysbios\sysbios.aem4f ..
gmake[2]: Nothing to be done for 'all'.

***** Build Finished *****
```

Physical Setup:



Debugging Process:



Successful Compilation:

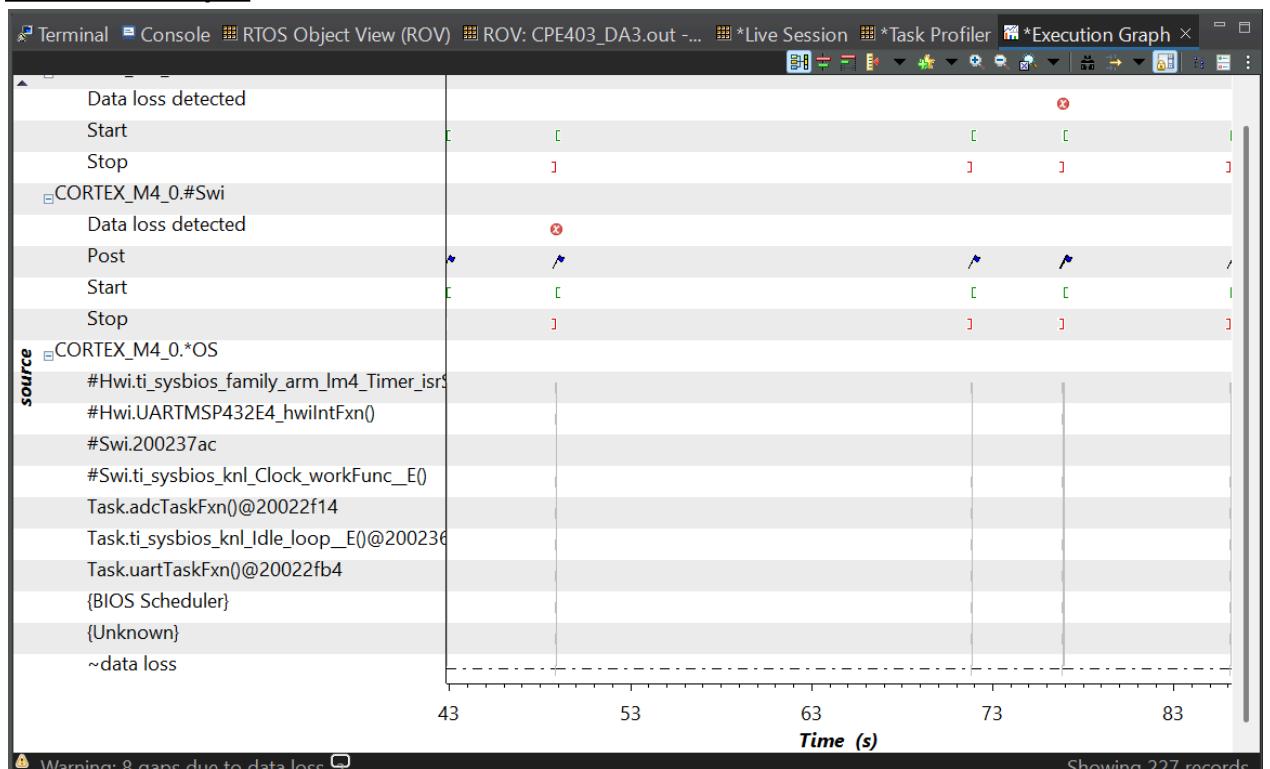
The screenshot shows the Code Composer Studio interface with the following details:

- Project Explorer:** Displays the project structure for "CPE403_DA3 [Active - Debug]". The structure includes:
 - Generated Source
 - Binaries
 - Includes
 - Debug
 - targetConfigs
 - MSP_EXP432E401Y_TIRTOS.cmd
 - mutex.c
 - src
 - Board.html
 - makefile.defs
 - mutex.cfg
 - mutex.syscfg
 - overview.rov.json
 - README.html
 - README.md
- Editor:** Shows the content of the file "Idle.c". The code is a header file for the Idle loop, containing comments about liability and the implementation of the Idle_loop function. It includes headers for std.h, sysbios/BIOS.h, sysbios/hal/Core.h, sysbios/hal/Hwi.h, and internal/Idle.xdc.h. It also includes MISRA.CPP.2012 rules.
- Console:** Displays the build logs for the "Debug" configuration of the "CPE403_DA3" project. The logs show the build command being run ("C:\ti\ccs1281\ccs\utils\bin\gmake -k -j 24 all"), the compilation of source files (src\sysbios\sysbios.aem4f), and the completion of the build process ("Build Finished").

RTOS Object Viewer:

RTOS Object View (ROV) - ROV: CPE403_DA3.out - CORTEX_M4_0												
Hwi Basic												
address	halHwiHandle	label	type	intNum	priority	group	subPriority	fxn	arg			
0x2002377c			Dispatched	35	224	7	0	ti_sysbios_family_arm_lm4_Timer_isrStub_E	0x20023488			
0x20023794			Dispatched	37	224	7	0	ti_sysbios_family_arm_lm4_Timer_isrStub_E	0x200234c8			
0x20000388			Dispatched	67	224	7	0	GPIO_hwIntFxn	0x8			
0x200003e8			Dispatched	21	224	7	0	UARTMSP432E4_hwIntFxn	0xe2e0			
Task Basic												
address	label		priority	mode	fxn		arg0	arg1	stackSize	stackBase	curCoreId	affinity
0x20023608	ti.sysbios.knl.Task.IdleTask	0	Running		ti_sysbios_knl_Idle_loop_E		0x0	0x0	512	0x200226e0	n/a	n/a
0x20022f14		3	Blocked		adcTaskFxn		0x0	0x0	2048	0x20020380	n/a	n/a
0x20022f64		2	Blocked		pwmTaskFxn		0x0	0x0	2048	0x20020b80	n/a	n/a
0x20022fb4		2	Blocked		uartTaskFxn		0x0	0x0	2048	0x20021380	n/a	n/a
Semaphore Basic												
address	label	event	eventId	mode	count	pendedTasks						
0x200230d4	none	n/a	binary	1	none							
0x20023130	none	n/a	counting	0	Fxn: adcTaskFxn, priority: 3, pendState: Waiting forever							
0x2002314c	none	n/a	counting	0	Fxn: pwmTaskFxn, priority: 2, pendState: Waiting forever							
0x200231b0	none	n/a	binary	0	none							
0x200231cc	none	n/a	counting	0	none							
0x20000368	none	n/a	binary	1	none							
0x200003a8	none	n/a	binary	1	none							
0x200003c8	none	n/a	binary	1	none							
0x20000408	none	n/a	binary	0	Fxn: uartTaskFxn, priority: 2, pendState: Waiting forever							
0x20000428	none	n/a	binary	0	none							
BIOS Module												
address	currentThreadType	rtsGateType		cpuFreqLow	cpuFreqHigh	clockEnabled	swiEnabled	taskEnabled	startFunc			
0x2002382c	Task	ti.sysbios.BIOS.GateMutex		120000000	0	true	true	true	0x9fed			

Execution Graph:



4. Declaration

I understand the Student Academic Misconduct Policy -
<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Joshua Martinez