CPE403 – Advanced Embedded Systems

# Design Assignment 1

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Joe Uesato

Email: uesatj1@unlv.nevada.edu

Github Repository link (root): https://github.com/joeuesato/lab\_submissions

Youtube Playlist: https://www.youtube.com/playlist?list=PLSBOvuRedzOf8JAhpVx0VsSteisJQUKv3

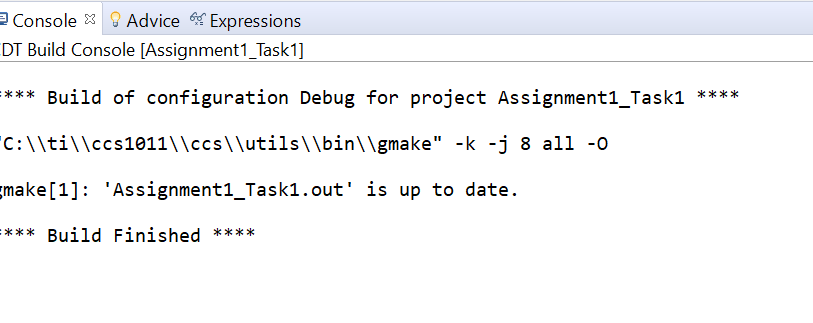
**Follow the submission guideline to be awarded points for this Assignment.**

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 the 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 CPE/403, Lastname, Firstname). Place all labs under the root folder TIVAC, 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 youtube videos as playlist under the name “cpe403”. The playlist should have the video sequence arranged as submission or due dates.
7. Only submit pdf documents. Do not forget to upload this document in the github repository and in the canvas submission portal.
8. Code for Tasks. for each task submit the modified 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 base code for the first task only. Use separate page for each task.
9. **#include** <stdint.h>
10. **#include** <stdbool.h>
11. **#include** <string.h>
12. **#include** "inc/tm4c123gh6pm.h" //def. for the interrupt and register assignments on the Tiva C Series device on the launchPad board
13. **#include** "inc/hw\_ints.h"
14. **#include** "inc/hw\_memmap.h"
15. **#include** "inc/hw\_adc.h"
16. **#include** "inc/hw\_types.h"
17. **#include** "inc/hw\_udma.h"
18. **#include** "driverlib/debug.h"
19. **#include** "driverlib/gpio.h"
20. **#include** "driverlib/interrupt.h"
21. **#include** "driverlib/pin\_map.h"
22. **#include** "driverlib/sysctl.h"
23. **#include** "driverlib/uart.h"
24. **#include** "driverlib/adc.h"
25. **#include** "driverlib/udma.h"
26. **#include** "driverlib/timer.h"
27. **#include** "driverlib/rom.h"
28. **#include** "driverlib/rom\_map.h"
29. **#include** "driverlib/systick.h"
30. **#include** "utils/uartstdio.h"
31. **#ifdef** DEBUG
32. void\_\_error\_\_(**char** \*pcFilename, uint32\_t ui32Line)
33. {
34. }
35. **#endif**
36. **#define** ADC\_SAMPLE\_BUF\_SIZE 1
37. **enum** BUFFERSTATUS
38. { *EMPTY*,
39. *FILLING*,
40. *FULL*
41. };
42. **#pragma** DATA\_ALIGN(ucControlTable, 1024)
43. uint8\_t ucControlTable[1024];
44. **static** **enum** BUFFERSTATUS BufferStatus[2];
45. **static** uint32\_t g\_ui32DMAErrCount = 0u;
46. **static** uint32\_t g\_ui32SysTickCount;
47. // Globals
48. uint32\_t ui32Period;
49. **char** buffer[4];
50. uint32\_t ui32ADC0Value[4];
51. **volatile** uint32\_t ui32TempAvg;
52. **volatile** uint32\_t ui32TempValueC;
53. **volatile** uint32\_t ui32TempValueF;
54. uint8\_t ui8LEDData=2; // For LED colors
55. **char** cmd; // For UART Input
56. uint32\_t ui32PinStatus = 0x00000000; // Variable to store the pin status of GPIO PortF
57. **void** **Timer1IntHandler**(**void**);
58. **void** **GPIOF0IntHandler**(**void**);
59. **void** **getTemp**(**void**);
60. **void** **LEDSetup**(**void**);
61. **void** **UARTinit**(**void**);
62. **void** **ADCinit**(**void**);
63. **void** **timerSetup**(**void**);
64. **void** **init\_DMA**(**void**);
65. **void** **uDMAErrorHandler**(**void**);
66. **int** **main**(**void**) {
67. // Configure Clock
68. **SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);
69. LEDSetup();
70. UARTinit();
71. ADCinit();
72. timerSetup();
73. // uDMA Setup
74. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UDMA); //Enable the clock to uDMA
75. MAP\_SysTickPeriodSet(**SysCtlClockGet**() / 100000u); //Sets the period of the SysTic counter to 10us
76. MAP\_SysTickIntEnable();
77. MAP\_SysTickEnable();
78. // Enable timer interrupts
79. **IntMasterEnable**();
80. // TimerEnable(TIMER1\_BASE, TIMER\_A); // Disable for tasks 2 and 3
81. **ADCSequenceEnable**(ADC0\_BASE, 2);
82. // Initial message to terminal display
83. // UARTprintf("Temperature:\n"); // Disable for tasks 2 and 3
84. **while** (1) // Wait forever
85. {
86. **UARTprintf**("Enter Command: ");
87. **char** cmd = **UARTgetc**();
88. **UARTprintf**("\n");
89. **switch** (cmd) {
90. **case** 'R':
91. **GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 2);
92. **break**;
93. **case** 'r':
94. **GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0);
95. **break**;
96. **case** 'G':
97. **GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 8);
98. **break**;
99. **case** 'g':
100. **GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0);
101. **break**;
102. **case** 'B':
103. **GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 4);
104. **break**;
105. **case** 'b':
106. **GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0);
107. **break**;
108. **case** 'T':
109. getTemp();
110. **UARTprintf**("C %3d\t\n",ui32TempValueC );
111. **break**;
112. **case** 't':
113. getTemp();
114. **UARTprintf**("F %3d\t\n",ui32TempValueF );
115. **break**;
116. **case** 'S':
117. ui32PinStatus = **GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);
118. **switch** (ui32PinStatus) {
119. **case** 0:
120. **UARTprintf**("RGB Status: Black (off)\n");
121. **break**;
122. **case** 2:
123. **UARTprintf**("RGB Status: Red\n");
124. **break**;
125. **case** 4:
126. **UARTprintf**("RGB Status: Blue\n");
127. **break**;
128. **case** 8:
129. **UARTprintf**("RGB Status: Green\n");
130. **break**;
131. **case** 6:
132. **UARTprintf**("RGB Status: Magenta\n");
133. **break**;
134. **case** 10:
135. **UARTprintf**("RGB Status: Yellow\n");
136. **break**;
137. **case** 12:
138. **UARTprintf**("RGB Status: Cyan\n");
139. **break**;
140. **case** 14:
141. **UARTprintf**("RGB Status: White\n");
142. **break**;
143. **default**:
144. **UARTprintf**("Invalid Color? There is an error.\n");
145. }
146. **break**;
147. **default**:
148. **UARTprintf**("Invalid Input. Please try again");
149. }
150. }
151. }
152. // Get temperature without the timer
153. **void** **getTemp**(**void**)
154. {
155. **ADCIntClear**(ADC0\_BASE, 2);
156. **ADCProcessorTrigger**(ADC0\_BASE, 2);
157. // ADCSequenceDataGet(ADC0\_BASE, 2, ui32ADC0Value);
158. ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
159. ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;
160. ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;
161. // Enable for another uDMA block transfer
162. **uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0), &ui32ADC0Value, ADC\_SAMPLE\_BUF\_SIZE);
163. **uDMAChannelEnable**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT); // Enables DMA channel so it can perform transfers
164. }
165. // Timer 1 ISR
166. **void** **Timer1IntHandler**(**void**)
167. {
168. // Clear the timer interrupt
169. **TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);
170. **ADCIntClear**(ADC0\_BASE, 2);
171. **ADCProcessorTrigger**(ADC0\_BASE, 2);
172. **ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);
173. ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
174. ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;
175. ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;
176. **UARTprintf**("C %3d\t",ui32TempValueC );
177. **UARTprintf**("F %3d\t",ui32TempValueF );
178. **UARTprintf**("\n");
179. }
180. **void** **GPIOF0IntHandler**(**void**) //interrupt handler for GPIO pin F0
181. {
182. //clear interrupt flag on pin F0
183. **GPIOIntClear**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);
184. // Update RGB color
185. **if**(ui8LEDData==8) {ui8LEDData=2;} **else** {ui8LEDData=ui8LEDData\*2;}
186. // Change color
187. **GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1| GPIO\_PIN\_2| GPIO\_PIN\_3, ui8LEDData);
188. **SysCtlDelay**(500000);
189. }
190. **void** **LEDSetup**(**void**)
191. {
192. //Port configuration (LEDS)
193. //Enable GPIOF port
194. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);
195. //set LEDS connected to pins as outputs
196. **GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);
197. //Unlock Pin F0 to use an interrupt on SW2
198. SYSCTL\_RCGC2\_R |= 0x00000020; // activate clock for Port F
199. GPIO\_PORTF\_LOCK\_R = 0x4C4F434B; // unlock GPIO Port F
200. GPIO\_PORTF\_CR\_R = 0x1F; // allow changes to PF4-0
201. // only PF0 needs to be unlocked, other bits can't be locked
202. GPIO\_PORTF\_AMSEL\_R = 0x00; // disable analog on PF
203. GPIO\_PORTF\_PCTL\_R = 0x00000000; // PCTL GPIO on PF4-0
204. GPIO\_PORTF\_DIR\_R = 0x0E; // PF4,PF0 in, PF3-1 out
205. GPIO\_PORTF\_AFSEL\_R = 0x00; // disable alt funct on PF7-0
206. GPIO\_PORTF\_PUR\_R = 0x11; // enable pull-up on PF0 and PF4
207. GPIO\_PORTF\_DEN\_R = 0x1F; // enable digital I/O on PF4-0
208. // Enable PF0 interrupt
209. // Register the interrupt handler for PF0
210. **GPIOIntRegister**(GPIO\_PORTF\_BASE, GPIOF0IntHandler);
211. //SW2 goes low when pressed
212. **GPIOIntTypeSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_FALLING\_EDGE);
213. //enable interrupts on PF0
214. **GPIOIntEnable**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);
215. // Start on Red LED
216. // GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, ui8LEDData); // Disable for tasks 2 and 3
217. }
218. **void** **UARTinit**(**void**)
219. {
220. // Configure peripherals
221. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);
222. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);
223. // Configure pins for UART
224. **GPIOPinConfigure**(GPIO\_PA0\_U0RX);
225. **GPIOPinConfigure**(GPIO\_PA1\_U0TX);
226. **GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);
227. **UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);
228. **UARTStdioConfig**(0, 115200, 16000000);
229. }
230. **void** **ADCinit**(**void**)
231. {
232. // Use for Tasks 1 and 2
233. /\* // Configure ADC
234. SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);
235. ADCHardwareOversampleConfigure(ADC0\_BASE, 32);
236. ADCSequenceConfigure(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0); // Changed to sequencer #2
237. ADCSequenceStepConfigure(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);
238. ADCSequenceStepConfigure(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);
239. ADCSequenceStepConfigure(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);
240. ADCSequenceStepConfigure(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);
241. ADCSequenceEnable(ADC0\_BASE, 2);
242. \*/
243. // Use for task 3
244. **GPIOPinTypeADC**(GPIO\_PORTE\_BASE, GPIO\_PIN\_2);
245. **SysCtlDelay**(80u);
246. // Use ADC0 sequence 0 to sample channel 0 once for each timer period
247. **ADCClockConfigSet**(ADC0\_BASE, ADC\_CLOCK\_SRC\_PIOSC | ADC\_CLOCK\_RATE\_HALF, 1);
248. **SysCtlDelay**(10); // Time for the clock configuration to set
249. **IntDisable**(INT\_ADC0SS0);
250. **ADCIntDisable**(ADC0\_BASE, 0u);
251. **ADCSequenceDisable**(ADC0\_BASE,0u);
252. // With sequence disabled, it is now safe to load the new configuration parameters
253. **ADCSequenceConfigure**(ADC0\_BASE, 0u, ADC\_TRIGGER\_TIMER, 0u);
254. **ADCSequenceStepConfigure**(ADC0\_BASE,0u,0u,ADC\_CTL\_CH0| ADC\_CTL\_END | ADC\_CTL\_IE);
255. **ADCSequenceEnable**(ADC0\_BASE,0u); //Once configuration is set, re-enable the sequencer
256. **ADCIntClear**(ADC0\_BASE,0u);
257. **ADCSequenceDMAEnable**(ADC0\_BASE,0);
258. **IntEnable**(INT\_ADC0SS0);
259. }
260. **void** **timerSetup**(**void**)
261. {
262. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); // Enabling Timer 1
263. // Configure Timer 1 module
264. **TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);
265. ui32Period = **SysCtlClockGet**()/2; // Period of 0.5s 2Hz
266. **TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1);
267. **IntEnable**(INT\_TIMER1A);
268. **TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);
269. }
270. **void** **uDMAErrorHandler**(**void**)
271. {
272. uint32\_t ui32Status;
273. ui32Status = MAP\_uDMAErrorStatusGet();
274. **if**(ui32Status)
275. {
276. MAP\_uDMAErrorStatusClear();
277. g\_ui32DMAErrCount++;
278. }
279. }
280. **void** **init\_DMA**()
281. {
282. **uDMAEnable**(); // Enables uDMA
283. **uDMAControlBaseSet**(ucControlTable);
284. **uDMAChannelAttributeDisable**(UDMA\_CHANNEL\_ADC0, UDMA\_ATTR\_ALTSELECT | UDMA\_ATTR\_HIGH\_PRIORITY | UDMA\_ATTR\_REQMASK);
285. **uDMAChannelAttributeEnable**(UDMA\_CHANNEL\_ADC0, UDMA\_ATTR\_USEBURST);
286. // Only allow burst transfers
287. **uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_SIZE\_16 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_16 | UDMA\_ARB\_1);
288. **uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_ALT\_SELECT, UDMA\_SIZE\_16 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_16 | UDMA\_ARB\_1);
289. **uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0), &ui32ADC0Value, ADC\_SAMPLE\_BUF\_SIZE);
290. **uDMAChannelEnable**(UDMA\_CHANNEL\_ADC0); // Enables DMA channel so it can perform transfers
291. }
292. **void** **SysTickIntHandler**(**void**)
293. {
294. // Update our system tick counter.
295. g\_ui32SysTickCount++;
296. }
297. **void** **ADCseq0Handler**()
298. {
299. **ADCIntClear**(ADC0\_BASE, 0);
300. **if** ((**uDMAChannelModeGet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT) == UDMA\_MODE\_STOP)
301. && (BufferStatus[0] == *FILLING*))
302. {
303. BufferStatus[0] = *FULL*;
304. BufferStatus[1] = *FILLING*;
305. }
306. **else** **if** ((**uDMAChannelModeGet**(UDMA\_CHANNEL\_ADC0 | UDMA\_ALT\_SELECT) == UDMA\_MODE\_STOP)
307. && (BufferStatus[1] == *FILLING*))
308. {
309. BufferStatus[0] = *FILLING*;
310. BufferStatus[1] = *FULL*;
311. }
312. }
313. Block diagram and/or Schematics showing the components, pins used, and interface.

TIVAC connected by USB directly to computer. No other connections.

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



See videos for code operation

1. Declaration

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

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

Joe Uesato