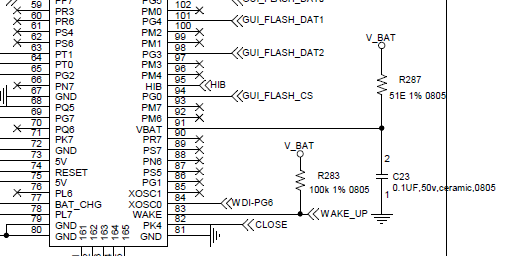
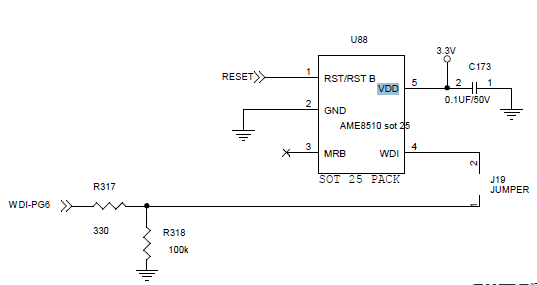
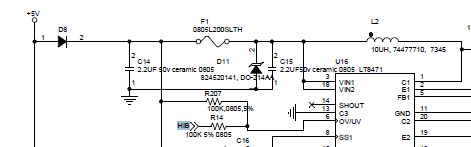
**Hibernation Details**

HW Schematic for hibernation is as given below:







Firmware application code is as mentioned below:

**apiHibernate.c -> This file has vfnInitHibernate() and HibEnterWakeExit() function definition.**

**vfnInitHibernate() is called once in main() and also from my application under HibEnterWakeExit().**

**HibEnterWakeExit() gets invoked once main power is OFF in my application.**

**Main.c -> Configured the clock as below:**

ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_16MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL |SYSCTL\_CFG\_VCO\_480),40000000);

**void** **vfnInitHibernate**()

{

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOL);

**GPIOPinTypeGPIOOutput**(GPIO\_PORTL\_BASE, PIN\_LOAD\_CONTROL);

**GPIOPinWrite**(GPIO\_PORTL\_BASE,PIN\_LOAD\_CONTROL,0);

// START: Configure PQ2 & Pk4 as input for Wake sources with respect to new hibernation board

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOQ);

**GPIOPinTypeGPIOInput**(GPIO\_PORTQ\_BASE, GPIO\_PIN\_2); // PQ2 for HANDWHEEL key

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOK);

**GPIOPinTypeGPIOInput**(GPIO\_PORTK\_BASE, GPIO\_PIN\_4); // PK4 for CLOSE key

// END: Configure PQ2 & Pk4 as input for Wake sources with respect to new hibernation board

//

// Need to enable the hibernation peripheral after wake/reset, before using

// it.

//

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_HIBERNATE);

//

// Wait for the Hibernate module to be ready.

//

**while**(!**SysCtlPeripheralReady**(SYSCTL\_PERIPH\_HIBERNATE))

{

}

}

**void** **HibEnterWakeExit**()

{

vfnInitHibernate();

//

// Initialize these variables before they are used.

//

ui32Status = 0;

ui32HibernateCount = 0;

ui32RTCWakeCount = 0;

ui32EXTWakeCount = 0;

ui32GPIOWakeCount = 0;

ui32RESETWakeCount = 0;

//

// Check to see if Hibernation module is already active, which could mean

// that the processor is waking from a hibernation.

//

**if**(**HibernateIsActive**())

{

//

// Read the status bits to see what caused the wake. Clear the wake

// source so that the device can be put into hibernation again.

//

ui32Status = **HibernateIntStatus**(0);

**HibernateIntClear**(ui32Status);

//

// Wake was due to RTC match.

//

**if**(ui32Status & HIBERNATE\_INT\_RTC\_MATCH\_0)

{

ui32RTCWakeCount++;

}

//

// Wake was due to Reset button.

//

**else** **if**(ui32Status & HIBERNATE\_INT\_RESET\_WAKE)

{

ui32RESETWakeCount++;

}

//

// Wake was due to the External Wake pin.

//

**else** **if**(ui32Status & HIBERNATE\_INT\_PIN\_WAKE)

{

ui32EXTWakeCount++;

}

//

// Wake was due to GPIO wake.

//

**else** **if**(ui32Status & HIBERNATE\_INT\_GPIO\_WAKE)

{

ui32GPIOWakeCount++;

}

//

// If the wake is due to any of the configured wake sources, then read

// the first location from the battery backed memory, as the

// hibernation count.

//

**if**(ui32Status & (HIBERNATE\_INT\_PIN\_WAKE | HIBERNATE\_INT\_RTC\_MATCH\_0 |

HIBERNATE\_INT\_GPIO\_WAKE | HIBERNATE\_INT\_RESET\_WAKE))

{

**HibernateDataGet**(&ui32HibernateCount, 1);

}

}

//

// Configure Hibernate module clock.

//

**HibernateEnableExpClk**(32768);

//

// If the wake was not due to the above sources, then it was a system

// reset.

//

**if**(!(ui32Status & (HIBERNATE\_INT\_PIN\_WAKE | HIBERNATE\_INT\_RTC\_MATCH\_0 |

HIBERNATE\_INT\_GPIO\_WAKE | HIBERNATE\_INT\_RESET\_WAKE)))

{

//

// Configure the module clock source.

//

**HibernateClockConfig**(HIBERNATE\_OSC\_LOWDRIVE);

}

//

// Enable RTC mode.

//

**HibernateRTCEnable**();

//

// Set the RTC to 0 or an initial value. The RTC can be set once when the

// system is initialized after the cold startup and then left to run. Or

// it can be initialized before every hibernate.

//

**HibernateRTCSet**(0);

//

// Set the match 0 register for 30 seconds from now.

//

**HibernateRTCMatchSet**(0, **HibernateRTCGet**() + MWG\_IDLE\_DELAY);

//

// Configure GPIOs used as Hibernate wake source - PK4

//

**GPIOPadConfigSet**(GPIO\_PORTK\_BASE, 0x10, GPIO\_STRENGTH\_2MA,

(GPIO\_PIN\_TYPE\_WAKE\_LOW | GPIO\_PIN\_TYPE\_STD\_WPU));

//

// Enable processor interrupts.

//

IntMasterEnable();

//

// If hibernation count is very large, it may be that there was already

// a value in the hibernate memory, so reset the count.

//

ui32HibernateCount = (ui32HibernateCount > 10000) ? 0 : ui32HibernateCount;

//

// Check if user wants to enter hibernation.

//

**if**(g\_bHibernate == **true**)

{

//

// Increment the hibernation count, and store it in the battery backed

// memory.

//

ui32HibernateCount++;

**HibernateDataSet**(&ui32HibernateCount, 1);

//

// Yes - Clear the flag.

//

g\_bHibernate = **false**;

//

// Read and clear any status bits that might have been set since

// last clearing them.

//

ui32Status = **HibernateIntStatus**(0);

**HibernateIntClear**(ui32Status);

**HibernateGPIORetentionDisable**();

**HibernateIntClear**(HIBERNATE\_INT\_PIN\_WAKE);

//

// Configure Hibernate wake sources.

//

**HibernateWakeSet**(HIBERNATE\_WAKE\_PIN | HIBERNATE\_WAKE\_GPIO |

HIBERNATE\_WAKE\_RESET | HIBERNATE\_WAKE\_RTC);

**GPIOPinWrite**(GPIO\_PORTL\_BASE,PIN\_LOAD\_CONTROL,PIN\_LOAD\_CONTROL);

**HibernateGPIORetentionEnable**();

//

// Request Hibernation.

//

**HibernateRequest**();

//

// Wait for a while for hibernate to activate. It should never get

// past this point.

//

**SysCtlDelay**(100);

//

// Wait here.

//

**while**(1)

{

}

}

}

**Observation**

****