Workshop Setup

The software/hardware used in this workshop are:

- **CCSv5.5.0.00077**

- **ControlSUITE v3.2.2** (installed to c:\ti)

- **F28069 Piccolo Experimenter Kit** (TMDSDOCK28069) with USB cable connected from Debug USB Port to PC

Notes:

1. If you are using different versions of software, some lab steps may require slight modifications and/or screenshots may look a bit different
2. If the labs are not being done in a class environment where the software is already installed, install the above listed software first
LAB conventions

- Lab steps are numbered for easier reference
  1. ...
  2. ...

- Explanations, notes, warnings are written in **blue**
  - Warnings are shown with !
  - Information is marked with i
  - Tips and answers are marked with 🧪
  - Questions are marked with ?
LAB 1: BLINKING LED EXAMPLE

30 MINUTES

Open CCS and select the default workspace
You can close the TI Resource Explorer View (it will not be used)
Refer to Lab handouts for instructions
Blinking LED Example: Exercise Summary

• Key Objectives
  – Import and build a simple program to blink LED
  – Start a debug session and load/flash the program to the controlCARD
  – Run the program to blink LED

• Tools and Concepts Covered
  – Workspaces
  – Welcome screen / Resource Explorer
  – Project concepts
  – Basics of working with views
  – Debug launch
  – Debug control
  – Profile Clock
  – Local History
  – Debugging existing code in flash (Loading Symbols)
  – Build Properties
  – Changing compiler versions
Import ‘BlinkingLED’ Project

1. Start up CCS and select a workspace (default one is fine)

2. Close Resource Explorer view

3. Import the BlinkingLED project by going to menu Project → Import Existing CCS Eclipse Projects

4. In the box Select search-directory, type or browse to the ControlSUITE directory: C:\ti\controlSUITE\development_kits\F28069 controlSTICK

5. Select the project “BlinkingLED”

6. Ensure check box Copy projects into workspace is enabled

7. Click Finish
Build ‘BlinkingLED’ Project

1. In the Project Explorer view, select the BlinkingLED project (it should appear [Active – Debug]) to make it active.

2. Click on the Build icon in the toolbar.

3. The Console view will appear at the bottom with build messages (information, warnings, errors) as the project builds.

4. The Problems view will also appear at the bottom to highlight any possible build errors or warnings.

5. If the build is successful, the Problems view will contain no errors (warnings may still be seen).

In this case, there may be a warning that this project was originally created using a version of the compiler that is currently not installed (this is ok to ignore).
Create a Target Configuration File

In order to connect to the board, a Target Configuration File must be created. This file contains the information about the JTAG emulator and the device to be connected.

1. In the **Project Explorer**, right-click on the project and select **New → Target Configuration File**

2. In the dialog box, type File name as `f28069_controlcard` and click **Finish**

3. The target configuration editor will be shown

4. For the **Connection** choose **Texas Instruments XDS100v1 USB Emulator**

5. For the **Board or Device** choose **Experimenter’s Kit – Piccolo F28069**

6. Click **Save**
Debugger is ready for launch

- The Target Configuration File will be added to the project.

- The [Active/Default] indicates this is the target configuration that will be used to debug this project (Active) and is also the Default for all other projects of the workspace (unless they have active configurations of their own).
Debug ‘BlinkingLED’ Project

1. Click on the “green bug” button – make sure the project is selected first!

   When you hit the green bug button, several actions are done automatically
   – Prompt to save modified source files
   – Build the project (incrementally)
   – Start the debugger (CCS will switch to the CCS Debug perspective)
   – Connect CCS to the target
   – Load the program on the target
   – Run to main()

2. Program should be loaded to target and CCS should be halted at main()
Blink LED

1. Press the *Run* button in *Debug* view to run the program. LED on the controlCARD should now be blinking.
Debugging: Using Watchpoints

A Watchpoint is a type of breakpoint that monitors activity on a memory address.

In this step we will set a watchpoint to halt the CPU every time the LED toggles.

1. **Press the** *halt/suspend* button to halt the running program
   - The code should stop somewhere in the *for()* loop

2. **Open the** *Breakpoints* view
   - Go to CCS menu View → Breakpoints

3. **Create a new** *Hardware Watchpoint*
   - *Location*: GpioDataRegs.GPBTOGGLE.bit.GPIO34
   - *Memory*: Write

4. **Click OK**
Debugging: Using Watchpoints

1. Run the target again several times

2. Execution will halt each time LED is toggled, (ie) whenever GpioDataRegs.GPBTOGGLE.bit.GPIO34 is written to
More Debugging (Review)

Investigate other debugging views (Open via View menu)
- Memory Browser
- Registers
- Disassembly (see next slide)

Set breakpoints
- Double click on a source line to set/clear
- See list of breakpoints with the Breakpoints view
- Note that the 28x only support two hardware breakpoints
  • A watchpoint uses a hardware breakpoint resource
**View: Disassembly**

1. Open the disassembly view by going to View->Disassembly
   - You can see the current location of the PC (small blue arrow) and any breakpoints (small blue circles)

2. Toggle the *Show Source* button. Note the toggling of interleaved source with the disassembly
Debugging: Using Profile Clock

The profile clock available on most devices and can be used to count cycles

1. Enable the Clock
   - Menu Run → Clock → Enable
   - The clock will now be displayed on the status bar

2. Click the Run button
   - Clock should now show ~40M cycles

Tip: Double-clicking on the clock icon will reset the count to ‘0’

3. Disable the previously set watchpoint
   - Go to Breakpoints view and uncheck the check box

Check that the watchpoint that is watching for writes to GpioDataRegs.GPBTOGGLE.bit.GPIO34 is enabled
Increase LED Blink Rate

The blink rate of LED can be changed by modifying the period of the Timer

1. Switch to the CCS Edit perspective

2. In the Project Explorer view, open BlinkingLED-Main.c

3. Modify line 83 of BlinkingLED-Main.c
   - From:  
   ```c
   CpuTimer0Regs.PRD.all = mSec500;// 500ms * 2(# of LED states) = 1s blink rate
   ```
   - To:  
   ```c
   CpuTimer0Regs.PRD.all = mSec100;// 100ms * 2(# of LED states) = 200ms blink rate
   ```

4. Save BlinkingLED-Main.c
View: Local History

CCS keeps a local history of source changes
Right-click on a file in the editor and select Team → Show Local History

You can compare your current source file against any previous version or replace it with any previous version
Double-click on a revision to open it in the editor
Right-click on a revision and select Compare Current with Local to compare that revision to the current version

CCS also keeps project history
Recover files deleted from the project
Right-click on the project and select Recover from Local History in the context menu
Rebuild Project

1. Make sure you are in CCS Edit perspective and project BlinkingLED is active

2. Rebuild the project
   - Menu Project → Build Project

   CCS will automatically detect that the program currently being debugged has changed/rebuilt and ask if it should reload the file

3. Select Yes and CCS will reload/reflash the program and run to main()
Blink LED

1. Switch to the *CCS Debug* perspective

2. Press the *Run* button 🔄 to run the program

- The LED on the controlCARD should now be blinking at a faster rate
- If it is not blinking, remember to **disable** (not delete) the watchpoint set in the previous session! 😊
Debugging: Measuring Clock Cycles

1. Pause execution of the program

2. Enable the watchpoint for ‘GpioDataRegs.GPBTOGGLE.bit.GPIO34’ by checking the box for it in the **Breakpoints** view

3. **Click the Run** button
   - Code should be halted at first write to GpioDataRegs.GPBTOGGLE.bit.GPIO34

4. **Double-click on the clock icon in the status bar to reset the value to ‘0’**

5. **Click the Run** button again
   - Clock should now show ~8M cycles (1/5 of 40M)
Terminate the Debug Session

1. In the *Breakpoints* view, **delete** (not disable) the watchpoint as it will not be used in the next lab – simply highlight it and press the <Delete> key.

2. In *Debug View*, click on the *Terminate* button
   This will terminate the debugger and return you to the *CCS Edit* perspective.
Loading Symbols for Flashed Program

If the program is already flashed to target (as it is in this case) and you just wish to debug the existing code flashed on the target, you can configure CCS to debug the project by loading symbols only.

1. Select the drop-down menu next to the ‘bug’ button and select ‘Debug Configurations.’
   - Debug Configurations - Cached information created when a debug session is first launched for a project or target configuration. Information cached includes which target configuration to use, debug settings…
Loading Symbols for Flashed Program

2. Select ‘BlinkingLED’ in the left panel and the Program tab in the right panel

3. Under Loading options, select Load symbols only

4. Then select Apply and then Debug
Loading Symbols for Flashed Program

The debugger will start up, connect to the target, and load only the symbols for the program for the debugger (no code is loaded/flashed on the target)

The program counter will be set to the entry point of the code and not at ‘main’

5. Go to menu Run->Go Main to run to main
   Program Counter will now be at main()

6. Terminate the debug session
Project Properties

1. Make sure you are in the *CCS Edit* perspective
2. Right click on the ‘BlinkingLED’ project and select *Properties*
3. Browse through General Settings and Compiler and Linker options
4. Click *Cancel* when finished
Update Compiler Tools

Internet connection is required to install updates using CCS update manager
Note: this compiler update will already be installed for hands-on workshops in class environment

1. Go to Help->Check for Updates
2. In the list of Available Updates, Deselect All and then select the latest version of C2800 Compiler Tools, in this case 6.2.3
Update Compiler Tools

3. Click *Next* and again *Next*

4. Accept terms of license agreement

5. Click *Finish*

6. Restart CCS when it prompts for restart and go through installation
Change Compiler Version

1. Make sure you are in the **CCS Edit** perspective
2. Right click on the ‘BlinkingLED’ project and select **Properties**
3. Notice that **Effective compiler version** is set to **TI v6.2.3**
   - CCS will default to use the latest version installed
4. Click on the **More…** button beside **Compiler version** **TI v5.2.2**
5. Notice the two compiler versions are available to select
6. Click **Cancel** to leave settings as-is
Optional: Change Build Options

Build options are set per build configuration

1. In the project properties, change the Configuration to F2806x_RAM

2. Change the optimization settings
   - Go to the Build → C2000 Compiler → Basic Options
   - Change the Optimization level to 3

3. Click OK
Optional: Change Build Configuration

1. **Change the active configuration of project to** *F2806x_RAM*
   - Right click on the Project
   - Select *Build Configurations ➔ Set Active ➔ F2806x_RAM*

2. **Build the project by clicking the build button**
   - In the console view you will see that the *F2806x_RAM* configuration has been built

---

You can also change the configuration and build it by clicking on the arrow beside the build button and selecting the configuration you want to build

- Select *F2806x_RAM* and it will build this configuration
- The active configuration is indicated by the Checkmark
Optional: Load the optimized code

1. Select the drop-down menu next to the ‘bug’ button and select *Debug Configurations*.

2. Select *BlinkingLED* in the left panel and the *Program* tab in the right panel.

3. Under *Loading options*, select *Load program*.

4. Select *Apply* and then *Debug*.

5. Press the *Run* button in *Debug* view to run the program. LED on the controlCARD should now be blinking.

6. Click the *Suspend* button to halt the program.

7. Click the *Terminate* button to terminate the debug session.
Blinking LED Example: Summary

- In this lab we completed the following:
  - Imported and built a simple program to blink the on-board LED
  - Started a debug session and flashed the program to the controlCARD
  - Ran the program to blink LED
  - Used data watchpoints to halt CPU during each LED toggle
  - Measured clock cycles
  - Studied Project Build Properties
  - Updated version of compiler tools
  - Changed compiler options and build configuration of the project and rebuilt/re-ran the code
LAB 2: CPU TIMER EXAMPLE (PORTABLE PROJECT)

30 MINUTES

Open CCS and select the default workspace, or continue working in existing workspace
Refer to Lab handouts for instructions
CPU Timer Example (Portable Project): Exercise Summary

• **Key Objectives**
  – Create a new portable project based on the “CPU Timer” example
  – Create workspace level variables for the project
  – Link files to the project using variables
  – Configure build properties using variables

• **Tools and Concepts Covered**
  – Portable Projects
  – Linked resources
  – Linked resource path variables
  – Build variables
Create a New Project

1. Launch the *New CCS Project* Wizard
   - Go to menu *File* → *New* → *CCS Project*

2. Fill in the fields as shown in the right
   - *Project name*: CPUTimer
   - *Device* → *Family*: C2000
   - *Variant*: Experimenter’s Kit – Piccolo F28069
   - *Connection*: Texas Instruments XDS100v1 USB Emulator

3. In *Project templates and examples*, select *Empty Project*

4. Click *Finish* when done
   *Generated project will appear in the Project Explorer view*
Create a Linked Resource Path Variable

Here we will create the Linked Resource Path Variable which will be used when linking source files (resources) to the project

1. Open the workspace preferences
   - Menu Window → Preferences

2. Go to the Linked Resources preferences
   - Type ‘Linked’ in the filter field to make it easier to find

3. Use the New button to create a ‘Linked Resource Variable’ (named controlSUITE) that points to the root location of controlSUITE directory
   - Specify the Name as controlSUITE and Location as C:\ti\controlSUITE

4. Click OK
Create a Build Variable

Here we will create the Build Variable which will be used when setting the project’s compiler and linker options

1. Go to the Build Variables preferences
   - Type ‘Variables’ in the filter field to make it easier to find

2. Use the Add button to create a ‘Build Variable’ (named controlSUITE) that points to the root location of the controlSUITE directory
   - Specify the Variable Name as controlSUITE, Type as Directory and Value as C:\ti\controlSUITE

3. Click OK and OK again
Link Source Files to Project

Here we will link source files relative to the Linked Resource Path Variable previously created

1. Open Windows Explorer and browse to:
   C:\ti\controlSUITE\device_support\f2806x\v136\F2806x_common\source

2. Select the following files and drag and drop them into the “CPUTimer” project in the CCS Project Explorer view
   - F2806x_CodeStartBranch.asm
   - F2806x_CpuTimers.c
   - F2806x_DefaultIsr.c
   - F2806x_PieCtrl.c
   - F2806x_PieVect.c
   - F2806x_SysCtrl.c
   - F2806x_usDelay.asm
Link Source Files to Project

A dialog will appear asking if you wish to Copy or Link the files:

1. Select *Link to files*

2. Select *Create link locations relative to: controlSUITE*
   - Use the new Linked Resource variable we created (*controlSUITE*)

3. Hit *OK*

Files will now appear in the Project Explorer with the ‘link’ icon
Link Source Files to Project

Follow the same procedure to link other source files

1. Drag and drop each of the following files one by one into the “CPUTimer” project in the CCS Project Explorer view
   - C:\ti\controlSUITE\device_support\f2806x\v136\F2806x_examples_ccsv5\cpu_timer\Example_2806xCpuTimer.c
   - C:\ti\controlSUITE\device_support\f2806x\v136\F2806x_headers\source\F2806x_GlobalVariableDefs.c
   - C:\ti\controlSUITE\device_support\f2806x\v136\F2806x_headers\cmd\F2806x_Headers_nonBIOS.cmd
   - C:\ti\controlSUITE\libs\math\IQmath\v160\lib\Iqmath_fpu32.lib
   - C:\ti\controlSUITE\libs\math\FPUfastRTS\V100\lib\rts2800_fpu32_fast_supplement.lib

2. In each case, in the dialog that appears, select Link to files

3. Select Create link locations relative to: controlSUITE
   - Use the new Linked Resource variable we created (controlSUITE)

4. Click OK
Link Files to Project

1. Right-click on source file “Example_2806xCpuTimer.c” and go to Properties. Click on Resource in the left pane. Notice how the Location parameter references the Linked Resource Path Variable.

2. Click Cancel
Modifying Project Properties

Here we will add include file search paths using the Build Variable

1. Right-click on the project and select Properties

2. In the Compiler Include Options, add the following entries to the list of include search paths:
   - `${controlSUITE}\device_support\f2806x\v136\F2806x_headers\include`
   - `${controlSUITE}\device_support\f2806x\v136\F2806x_common\include`
   - `${controlSUITE}\libs\math\IQmath\v160\include`
   - `${controlSUITE}\libs\math\FPUfastRTS\V100\include`

   ‘${<BUILD VARIABLE>}’ is the syntax to use a Build Variable in the project properties.

   Linked Resource Path Variables are only used when linking source files to a project. They cannot be used for build options. Use Build Variables when modifying build options.
Project Properties

1. Go to Resource → Linked Resources to see all the Linked Resource Path Variables that are available to the project
   This will show all variables created at the project level and workspace level

2. Check that the workspace level Linked Resource Path Variable that was created appears in the list
   Variables may be edited here but changes will only be recorded at the project level (stored in the project files)
Project Properties

The **Linked Resources** tab will show all the files that have been linked to the project.
- It will sort them by files linked with a variable and files linked with an absolute path.

Links can be modified here with the **Edit…** button.

Links can be converted to use an absolute path with the **Convert…** button.
1. Go to Build → Variables tab to see all the Build Variables that are available to the project. Only project level variables will be listed by default.

2. Enable the Show system variables checkbox to see variables set at the workspace and system level. See how the workspace level Build Variable that was created earlier (controlSUITE) appears in the list.

3. Click OK
Build the Program

1. Make sure the ‘CPUTimer’ project is in focus

2. Right-click on the project and select Build Project
   The project should build without errors
CPU Timer Example (Portable Project): Summary

• In this lab we completed the following:
  – Created a new portable project
  – Created workspace level variables for the project – Linked Resource path variable and Build variable
  – Linked source files to the project using Linked Resource path variable
  – Configured include paths for compiler using Build variable
  – Validated that project builds without errors
LAB 3: REAL-TIME INTERRUPTS EXAMPLE: (REAL-TIME DEBUG)

30 MINUTES

Open CCS and select the default workspace, or continue working in existing workspace
Refer to Lab handouts for instructions
Real-time Interrupts Example (Real-time Debug): Exercise Summary

• Key Objectives
  – Demonstrate how to access/modify memory and registers in real time when processor is running
  – Enable real-time mode in Code Composer Studio
  – Set up interrupts in code so they can continue to run when debugger is halted
  – Step through code while time critical interrupts continue to be serviced

• Tools and Concepts Covered
  – Real-time access to memory and registers
  – Real-time mode and real-time interrupts
Import Example Project

1. Go to the menu Project ➔ Import Existing CCS Eclipse Project

2. Click the Browse button and browse to:
   - C:\ti\controlSUITE\device_support\f2806x\v136\F2806x_examples_ccsv5\epwm_real-time_interrupts

3. Click OK

4. Select the project Example_2806xEPwmRealTimeInt

5. Ensure the check box for Copy projects into workspace is selected

6. Click Finish
Review code

1. In *Project Explorer* view, expand the project *Example_2806x_EPwmRealTimeInt*

2. Open the source file *Example_2806xEPwmRealTimeInt.c*

   Review the code and observe the following:
   - The program configures ePWM1 timer and increments a counter (EPwm1TimerIntCount) each time interrupt is taken
   - LED2 on the controlCARD is toggled in the main routine and LED3 is toggled in the interrupt service routine epwm1_timer_isr
   - FREE_SOFT bits in EPwm1Regs and DBGIER.INT3 bit are initially cleared. These bits must be set to designate ePWM1 interrupt as time critical and operational in real time mode after halt command. During the lab, these bits will be set through the *Expressions* view in CCS when processor is halted to see how real-time interrupts work
Important Variables

While running the labs, the following variables are of interest. Observe how they change while the program is running and halted:

- **EPwm1TimerIntCount** - counts the number of times the ISR for PWM1 is serviced
- **EPwm1Regs.TBCTR** - register containing the counter for PWM1 – observe that it will stop with an emulation halt unless FREE_SOFT bits are set
- **EPwm1Regs.TBCTL.bit.FREE_SOFT** - 2 register bits that establish how the counter for the PWM reacts to an emulation halt command. The default 00 in these bits will stop the counter when a halt command is received. A 10 or 11 will allow the counter to run freely after the halt command is received
- **DBGIER.INT3** - Debug Interrupt Enable Register bit that is used to designate whether the corresponding interrupt is time critical. Setting this bit will designate the PWM interrupt to be time critical and operate as real-time interrupt
Build the Example Project

1. In the Project Explorer view, right-click on Example_2806xEPwmRealTimeInt and select Build Project
Create a Target Configuration File

1. Right-click on the project and go to New → Target Configuration File
2. Specify f28069_controlcard as the name
3. Click Finish and the Target Configuration Editor will open
4. For the Connection, select Texas Instruments XDS100v1 USB Emulator
5. Type F28069 in the Board or Device filter field and then select Experimenters’s Kit – Piccolo F28069
6. Click Save
Load the Example program

1. Select the *Example_2806xEPwnRealTimeInt* project in the *Project Explorer* view

2. Click on the “green bug” button to debug the project

3. Code should be loaded to target and halted at “main”
Watch Variables and Setup Continuous Refresh

1. Go to menu View->Memory Browser

2. Enter &EPwm1TimerIntCount in the address field of Memory Browser to view the address of variable “EPwm1TimerIntCount”

3. Arrange the views such that the Expressions, Memory Browser and Registers views are all visible at once

4. Add these variables to the Expressions view:
   – EPwm1TimerIntCount
   – EPwm1Regs.TBCTR
   – EPwm1Regs.TBCTL.bit.FREE_SOFT
   – DBGIER.INT3

5. In Registers view, expand the ePWM1 register

6. Enable Continuous Refresh on all three views
Run the C28x

1. **Click the Resume button to run the code**
   
   - Both LEDs on the board should be blinking, LD2 (toggled in main routine) and LD3 (toggled in ISR routine). LD2 appears to stay on continuously but is actually just blinking at a very high speed. The *Expressions, Memory Browser* and *Registers* view update in real-time while the processor is running.

2. **Halt C28xx core by clicking the Suspend button**
   
   - The LEDs on the board stop blinking and the views stop updating.
Enable Real-time Mode

1. Enable real-time mode in CCS by clicking the button shown here:

2. Click Resume to run the core.

3. In the Expressions view, set $EPwm1Regs.TBCTL.bit.FREE_SOFT$ bit to 2 to enable free run mode on the counter
   - Setting this bit allows the timer to continue to increment even when the target is halted

4. Click the Suspend button to halt the core
   - Notice the EPwm1Regs.TBCTR keeps changing in the Expressions view as it is set to free run

However the interrupt is not being serviced (EPwm1TimerIntCount is not incrementing and LD3 has also stopped blinking) as it has not been designated as a real-time interrupt
Real-time Interrupts

1. In the *Expressions* view, set `DBGIER.INT3` to be 1 to designate PWM1 as a real-time interrupt
   - Notice that EPwm1TimerIntCount now continues to increment even though we are halted. Similarly notice that LD3 which is toggled within ISR has resumed toggling while LD2 which is toggled within main routine remains stopped. This confirms that the interrupt is being serviced even when we are halted.

2. Click the *Step Into* button several times to single-step through the code
   - Notice that LD2 toggles on and off as you step through the main routine while LD3 continues to blink.

   *These steps demonstrate that, in real-time mode, the PWM1 interrupt that we designated as time critical is continuing to be serviced even when the core is halted.*

3. Click the *Terminate* button to terminate the debug session
Summary

• In this lab we completed the following:
  – Accessed memory and registers in real time while processor was running
  – Enabled real-time mode in CCS
  – Set up interrupts so they can continue to run when debugger has halted the processor
  – Stepped through code while time critical interrupts continue to be serviced