Cortex-M4 Experiments with S800 Board

Experiment One: GPIOs and Clock Control

■ General Description and Goals:

In this experiment, you should be familiar with: 1) KEIL μVision5 IDE and how to create new projects; 2) clock sources of TM4C1294NCPDT MCU and the power consumption when different clock sources are used; 3) GPIO programming in two programming modes, i.e., directly accessing registers and using the TivaWare™ Peripheral Driver Library. You are free to choose any one of the three methods in all future experiments.

From the JBox site, you can download the directory about the first experiment, containing the example codes to use GPIO PF0 as output to control LED D4 and GPIO PJ0 as input to read key press on USR_SW1 on the LaunchPad. You need to read the example codes, compile and make the target, download the target to your experiment board, run your program, and check with the result.

■ Introduction to programming with KEIL μVision5 IDE:

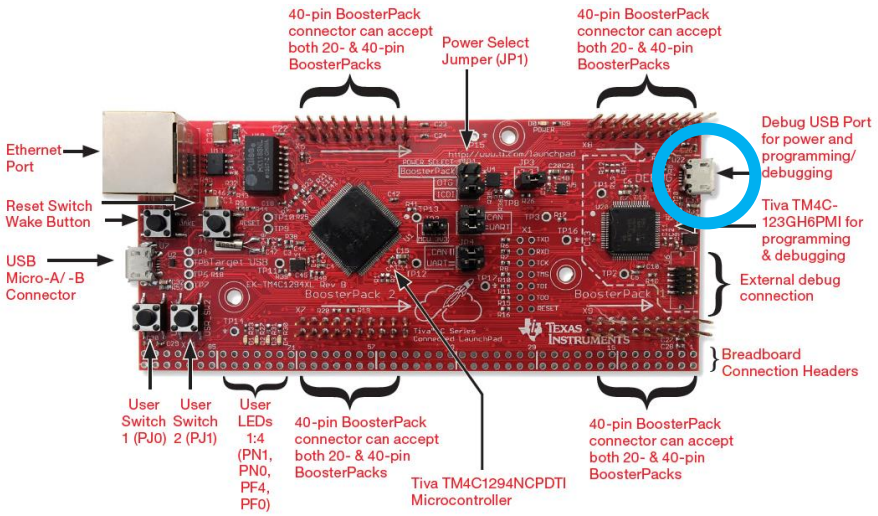
1. Software Installation:

Install the following software on your computer running Windows 7 or above version with 2G memory:

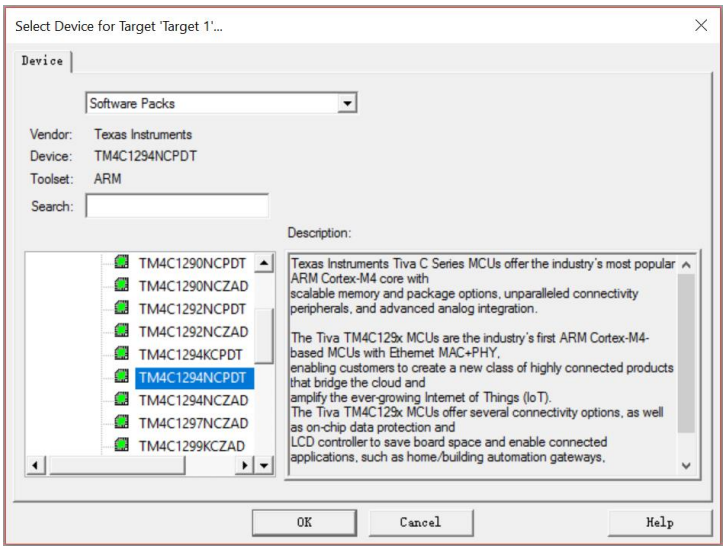
a)	MDK522	*the main setup file of KEIL 5.22*
b)	MDKCM522	*the setup file for KEIL legacy devices*
c)	Keil.TM4C_DFP.1.1.0	*Tiva C Series device support and examples*
d)	SW-EK-TM4C1294XL-2.1.4.178	*firmware development package setup*
e)	uniflash_sl.4.1.1250	*UniFlash setup*

2. Setup Programming Environment:

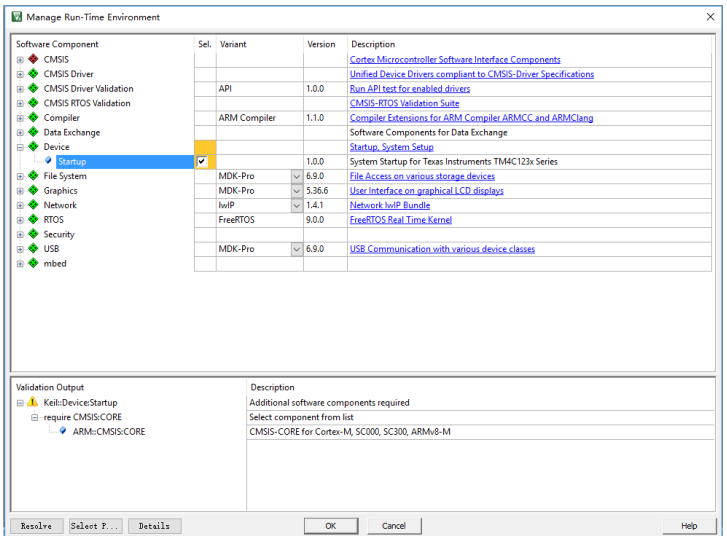
First, connect your S800 evaluation board through the ICDI with your computer via a Micro-USB cable.



Second, create a directory for a new project, for example, `C:\S800\EXP1`, then run KEIL μVision5 IDE, from the [Project] menu select [New μVision Project], select the project directory `C:\S800\EXP1`, and name your project, e.g., "`exp1.uvprojx`". Then, select the target MCU TM4C1294NCPDT.

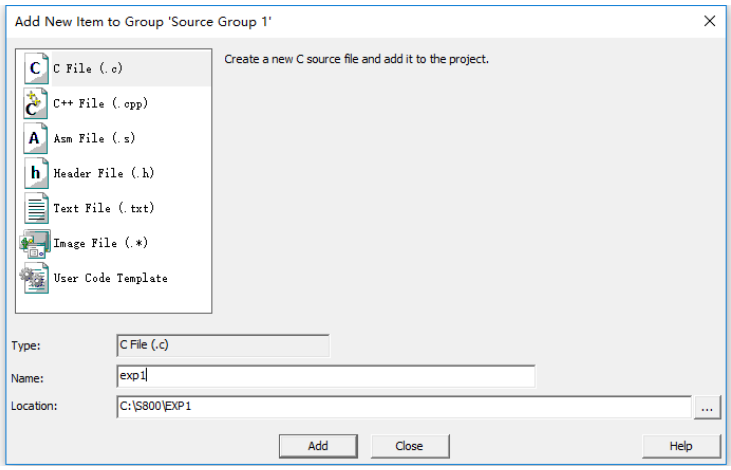


In the next dialogue, check Startup in Device to let KEIL generate Startup code.



After that, you should see a new project in the Project Window, containing a "`Source Group 1`" directory and a "`Device`" directory with two files, i.e., `Startup_TM4C129.s` and `System_TM4C129.c`.

Right click on the "`Source Group 1`" directory, select [Add New Item to Group "Source Group 1"], and generate a C source file, e.g., `exp1.c`.



Double click `exp1.c` to edit the file and copy the following example code:

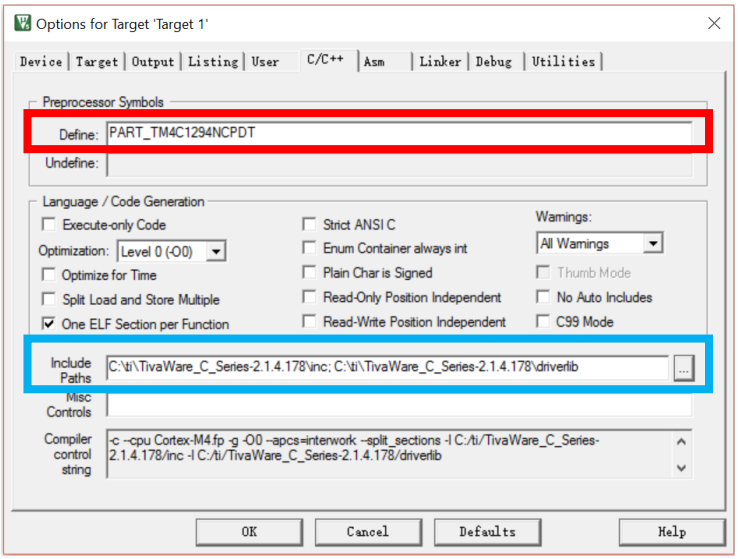
```
#include <stdint.h>
#include <stdbool.h>
#include "hw_memmap.h"
#include "debug.h"
#include "gpio.h"
#include "hw_types.h"
#include "pin_map.h"
#include "sysctl.h"

int main(void)
{
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
	//Enable PortF
	while (1);
}
```

Now, you can try to build your project from the [Project] menu select [Build Target] and you should see errors. The reason is that the included head files cannot be found at the current directory.

To fix this, specify the paths for C compiler to find those head files. Specifically, right click on “Target 1” and select [Option for Target “Target 1”]. In the “C/C++” tab, specify the paths to the required head files, separated by “;”. In this example, the paths should be:

```
C:\ti\TivaWare_C_Series-2.1.4.178\inc;
C:\ti\TivaWare_C_Series-2.1.4.178\driverlib
```



In addition, in the "C/C++" tab, we predefine a symbol "`PART_TM4C1294NCPDT`" which is checked in "`pin_map.h`" to specify GPIO Port/Pin mapping definitions for our target MCU TM4C1294NCPDT. Alternatively, you can add "`#define PART_TM4C1294NCPDT`" at the beginning of your main C source file as in the following example:

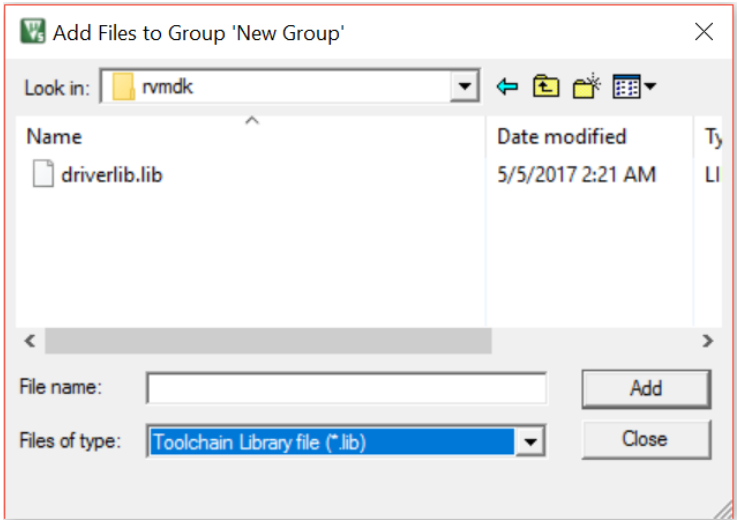
```
#define PART_TM4C1294NCPDT
#include <stdint.h>
#include <stdbool.h>
#include "hw_memmap.h"
#include "debug.h"
#include "gpio.h"
#include "hw_types.h"
#include "pin_map.h"
#include "sysctl.h"

int main(void)
{
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        //Enable PortF
        while (1);
}
```

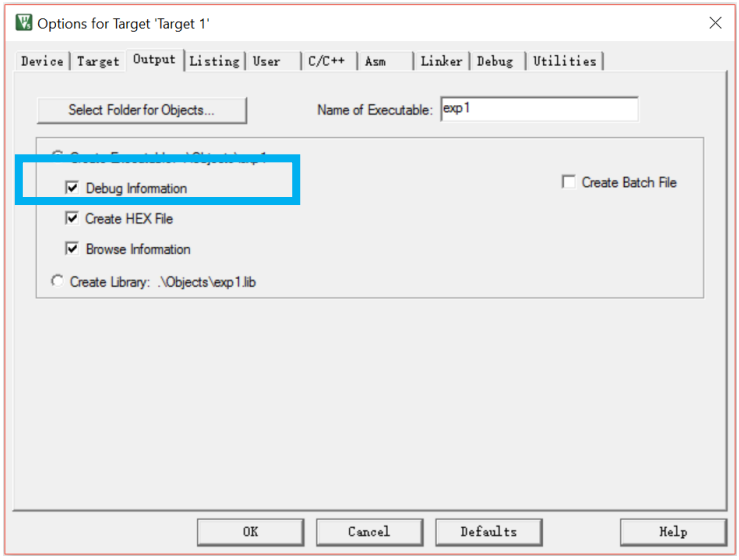
Now, you can try to build your project again and you should see new error "Undefined symbol SysCtlPeripheralEnable (referred from exp1.o)". The reason is that the executable code for the function `SysCtlPeripheralEnable` cannot be found for the linker program to generate final target executable file.

To fix this, you should add the TivaWare™ Peripheral Driver Library file into your

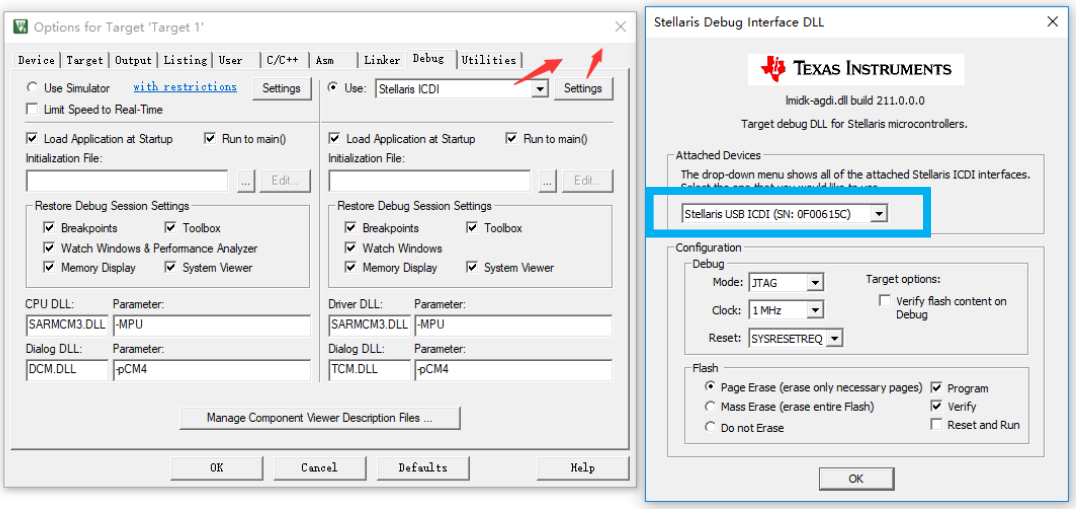
project. In specific, right click on “Target 1” and select [Add Group]. Then, right click “`New Group`”, select [Add Existing File to Group “New Group”], and add “`C:\ti\TivaWare_C_Series-2.1.4.178\driverlib\rvmdk\driverlib.lib`” (note the file type is .lib).



Now, you can try to build your project again and you should see no error. To execute your compiled program, you should download the program to the flash of the MCU on the LaunchPad. For this purpose, right click on “Target 1” and select [Option for Target “Target 1”]. In the “Output” tab, check “Create HEX file” and in the “Debug” tab, select to use “Stellaris ICDI”.



Click on the “settings” button, you should see the following figure



Finally, you can download the compiled program to MCU from [Falsh] menu and select [download].

■ Experiment Requirements:

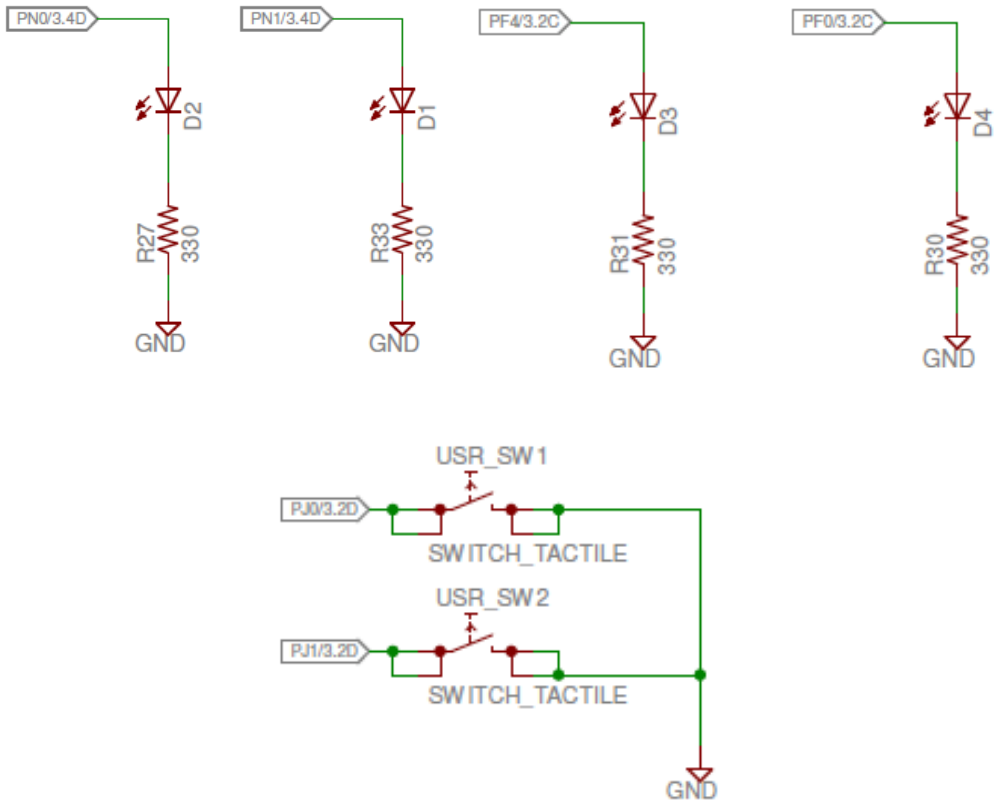
1. Create a project, add the provided `exp1.c`, build the project, download the compiled program to the MCU, and reset the MCU to run the program on MCU. You should see that LED D4 flashes and when pressing USR_SW1 it flashes faster.
2. Program to change the clock source used in your program including the 16 MHz PIOSC, the 25 MHz MOSC, and the PLL, respectively, and see the changes of LED D4 and the current values shown on the meter.
3. Program to achieve the task: when USR_SW1 is pressed, LED_M0 is turned on, and when USR_SW1 is released, LED_M0 is turned off; when USR_SW2 is pressed, LED_M1 is turned on and when USR_SW2 is released, LED_M1 is turned off.
4. Program to achieve the task: when USR_SW1 is pressed in the first time, LED_M0 flashes; when USR_SW1 is pressed in the second time, LED_M0 is turn off; when USR_SW1 is pressed in the third time, LED_M1 flashes; when USR_SW1 is pressed in the fourth time, LED_M1 is turn off; the process repeats.

■ Related On-Board Components:

On the LaunchPad:

Label	Connected MCU Pin	Description
RESET	RESET	TM4C1294NCPDT Reset Button: low effective
WAKE	WAKE	Wake event for bring the MCU back from hibernation mode
USR_SW1	PJ0	User button: low effective
USR_SW2	PJ1	User button: low effective
D0		3.3V power indicator, green LED, high effective
D1	PN1	User green LED, high effective

D2	PN0	User green LED, high effective
D3	PF4	User green LED, high effective
D4	PF0	User green LED, high effective



Note that PJ0 and PJ1 connected to USR_SW1 and USR_SW2, respectively, should be programmed with weak pull-up configuration in order to get stable button status.

On the blue board:

Label	Connected MCU Pin	Description
LED_M0	PF0	User green LED, low effective
LED_M1	PF1	User green LED, low effective
LED_M2	PF2	User green LED, low effective
LED_M3	PF3	User green LED, low effective
D10		3.3V power indicator, red LED, high effective

■ Related Peripheral Driver Library Functions:

GPIO:

1) To program on a GPIO port, first you need to supply system clock to that GPIO block using `SysCtlPeripheralEnable();`
2) You can check whether a peripheral is ready after clocking: `SysCtlPeripheralReady();`
3) To configures pin(s) for use as GPIO inputs, use

`GPIOPinTypeGPIOInput()` and for use as GPIO outputs, use `GPIOPinTypeGPIOOutput()`; Sets the pad configuration for the specified pin(s), use `GPIOPadConfigSet()`; read a value from or write a value to the specified pin(s) use `GPIOPinRead()` and `GPIOPinWrite()`, respectively;

4) To configure the alternate function of a GPIO pin, use: `GPIOPinConfigure()` and to fully configure a pin, a `GPIOPinType*()` function should also be called.

Code Example:

1) Definitions:

gpio.h

```
#define GPIO_PIN_0  0x00000001  // GPIO pin 0
……
#define GPIO_PIN_7  0x00000080  // GPIO pin 7
```

hw_memmap.h

```
#define GPIO_PORTF_BASE   0x40025000  // GPIO Port F
……
```

2) GPIO pin configuration:

```
//Enable PortF
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
//Wait for the GPIO port F ready
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));
// Set PF0 as Output pin
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);
//Set the PJ0,PJ1 as input pin
GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE,GPIO_PIN_0 |
GPIO_PIN_1);
//Set the PJ0,PJ1 has pull-up resistor
GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0 |
GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
```

System Clock:

1) To configure the system clock: `SysCtlClockFreqSet();`
2) **For TM4C129 devices**, the return value from `SysCtlClockFreqSet()` indicates the system clock frequency.

Code Example:

```
//Use PLL to generate frequency of 480MHz, and divide the PLL frequency to
get system clock of 20MHz, the divisor must be an integer
SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |SYSCTL_OSC_MAIN
|SYSCTL_USE_PLL |SYSCTL_CFG_VCO_480), 20000000);
//Use MOSC as the system clock of 25MHz
SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |SYSCTL_OSC_MAIN
|SYSCTL_USE_OSC), 25000000);
```

■ Questions:

1) The MCU has an internal 16MHz PIOSC and 25MHz MOSC. What if you specify a wrong frequency parameter when calling `SysCtlClockFreqSet()`? For instance, `SYSCTL_OSC_MAIN | SYSCTL_XTAL_20MHZ`?

2) Can you set a system clock frequency, which is lower than the MOSC? For example, when MOSC is 25MHz, can the system clock be only 20MHz?

3) When setting the maximum system clock frequency, i.e., 120MHz, with PLL, what will happen to the flashing LED and why?

4) Explain the meaning of each parameter in the follwing function call `GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_PIN_0)`.

5) Why do we need to program PJ0 and PJ1 with weak pull-up configuration?