Cortex-M4 Experiments with S800 Board

Experiment Two: I²C and SysTick Interrupts

■ General Description and Goals:

In this experiment, you should be familiar with: 1) I²C protocol; 2) PCA9557 and TCA6424 I²C I/O expender chips; 3) LEDs and digitron display; 4) interrupt driven programming.

■ Experiment Requirements:

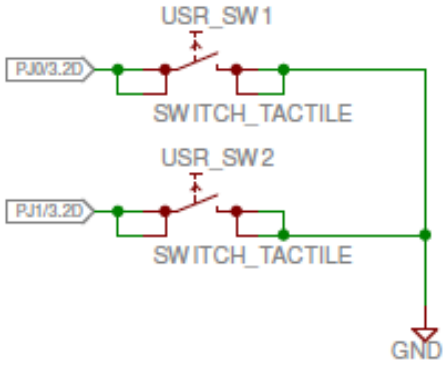
1. Read the example code and understand the initialization of PCA9557 and TCA6424C chips. In this example code, PCA9557 is used control LEDs and TCA6424 is used to control the 8-digit digitron.
2. Program to control LEDs and digitron to display in a moving fashion. In specific, when LED1 is turned on, the first digit of the digitron displays "1". Then, LED1 and the first digit are put off and the second LED LED2 is turned on and the second digit of the digitron displays "2". The process continues until the LED8 is turn on and the eighth digit of the digitron displays "8". After that, the whole process repeats.
3. In addition to Requirement 2, when USR_SW1 is pressed, the movement of the LEDs and the digitron stops and when USR_SW1 is released, the process continues.
4. Use interrupt-driven programming on SysTick to achieve Requirement 2 and 3, respectively.

5.* Use SysTick to simulate a multi-task system where Task1: PF0 flashes; Task 2: moving display of LEDs and the digitron; Task 3: press and hold USR_SW1 to keep PN0 on, and put off PN0 when USR_SW1 is released. When Task 3 is running, Task 1 and Task 2 are suspended; otherwise, Task 1 and Task 2 run alternately.

■ Related On-Board Components:

On the LaunchPad:

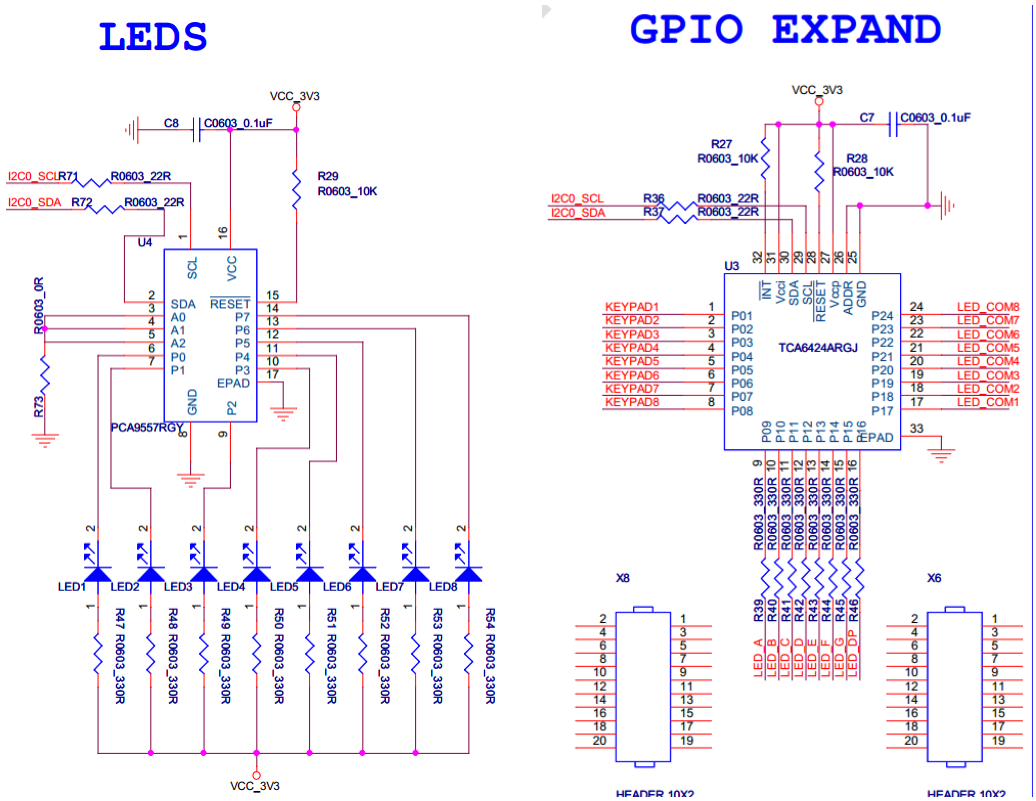
Label	Connected MCU Pin	Description
RESET	RESET	TM4C1294NCPDT Reset Button: low effective
WAKE	WAKE	Wake event for bring the MCU back from hibernation mode
USR_SW1	PJ0	User button: low effective
USR_SW2	PJ1	User button: low effective
D0		3.3V power indicator, green LED, high effective
D1	PN1	User green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

System Clock:

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

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.

SysTick:

1) To setup the initial count of SysTick: `SysTickPeriodSet();`
2) To enable SysTick: `SysTickEnable();`
3) **For software pulling**, you can read the current count of SysTick: `SysTickValueGet();`
4) **For interrupt-driven programming**, first you need to enable SysTick exceptions: `SysTickIntEnable()`; then enable the processor interrupt: `IntMasterEnable()`; last, implement the SysTick handler.

I^2C:

1) To program on a I^2C module, first you need to supply system clock to that I^2C module using `SysCtlPeripheralEnable();`
2) If pins used by the I2C module is used as GPIO by default, then you need to configure these pins to the proper function;
3) Enable the I^2C Master function of the I^2C module using `I2CMasterEnable();`
4) Set the desired SCL clock speed using `I2CMasterInitExpClk();`
5) Specify the slave address of the master: `I2CMasterSlaveAddrSet();`
6) **For master writing**, place data (byte) to be transmitted in the data register: `I2CMasterDataPut()`;start to transfer data using `I2CMasterControl();` you can check whether the data transfer is completed using `I2CMasterBusy();` repeat this process until all data is transferred (putting an end condition when calling `I2CMasterControl()`);

7) **For master reading**, start to receive data using `I2CMasterControl();` you can check whether the data transfer is completed using `I2CMasterBusBusy();` get the value from the return of `I2CMasterDataGet();`

■ Questions:

TBD.