


# Cortex-M4 Experiments with S800 Board

## Experiment Three: UART and Interrupts

### ■ General Description and Goals:

In this experiment, you should be familiar with: 1) serial communication with UART; 2) Blocking and non-blocking operations of UART; 3) interrupt-driven programming and the prioritization of interrupts.

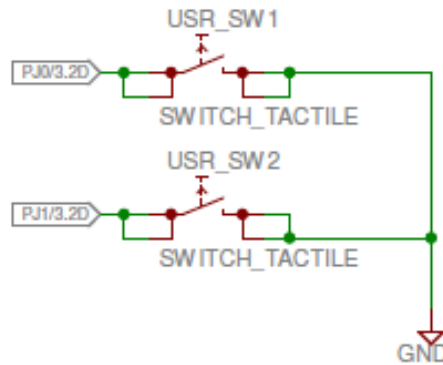
### ■ Experiment Requirements:

1. Read the example code in file `exp3-1.c` and understand the initialization of UART0. After the initialization, the S800 board sends a string of “HELLO, WORLD!” to the host (a PC running a serial communication software, such as ).
2. Based on the above requirement, read the so-called “UART ECHO” code in file `exp3-2.c`, in which the host first sends a string to the S800 and the S800 returns the received string back to the host.
3. Modify the code in Requirement 2 to use interrupts of UART0 to transmit and receive data via non-blocking operations. In addition, when the UART0 is receiving data, LED connected to PN1 will be turned on. See example code in `exp3-3.c`.
4. Program to implement AT comments: 1) when string “AT+CLASS” is sent from the host, the S800 returns “CLASS#####”, where ##### is your class ID; 2) when string “AT+STUDENTCODE” is send, the S800 returns “CODE\*\*\*\*\*”, where \*\*\*\*\* is your student ID.
5. Modify the above code to accept both uppercase and lowercase strings.
6. Program to implement a digital clock:
  - 1) the clock automatically runs on every 1s after reset;
  - 2) the host can send a “SET” command, such as “SET12:56:03” or “12-56-03” to the S800 and when received, the S800 sets its clock to this time and returns the adjusted time (in the form of “TIME12:56:03”) back to the host;
  - 3) the host can send a “INC” command, such as “INC00:00:12” to the S800 and when received, the S800 increases its clock time by 12 seconds and returns the adjusted time back to the host;
  - 4) the host can send “GETTIME” command to the S800 and the S800 will returns back the current clock time back to the host.
7. Interrupt priority experiment. Based on Requirement 3, change the priority level of UART0 to preempt SYSTICK. In this case, when USR\_SW2 is pressed and held and the host is sending a message, the MCU will run and been stuck in the handler of UART0. If USR\_SW1 is pressed, LED D2 has no response. See example code in `exp3-4.c`.  Modify the code to make SysTick have higher priority than UART0 and check the observation when the MCU is in the handler of UART0 and USR\_SW1 is pressed.

## ■ Related On-Board Components:

On the LaunchPad:

Label	Connected MCU Pin	Description
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

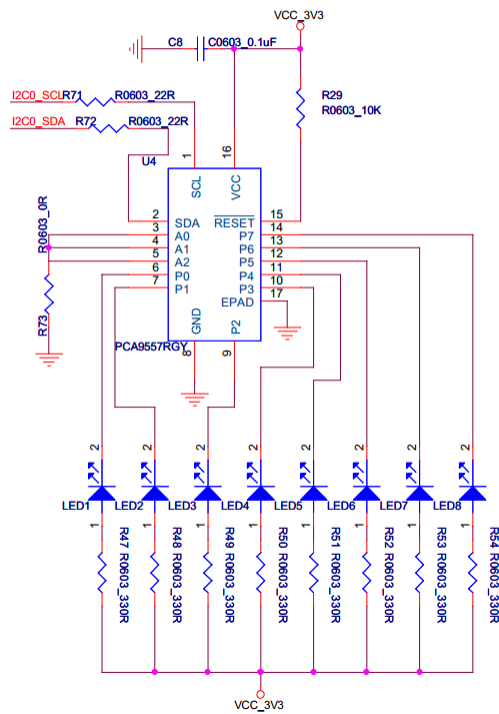


*Note that PJ0 and PJ1 connected to USR\_SW1 and USR\_SW2, respectively, should be programmed with weak pull-up configuration in order to get stable button status.*

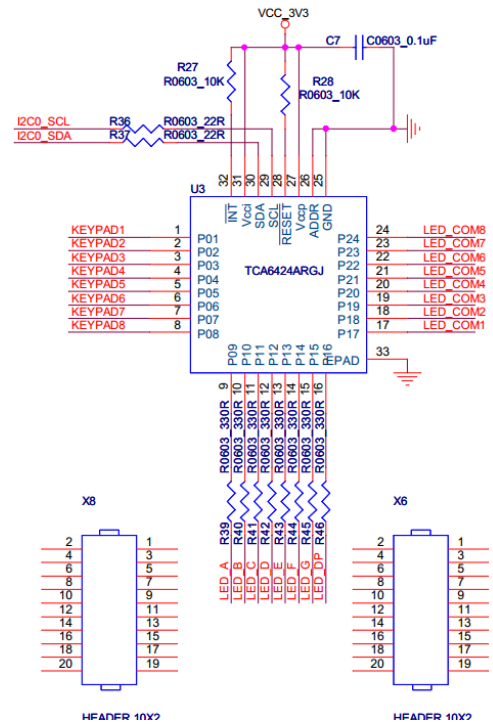
On the blue board:

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

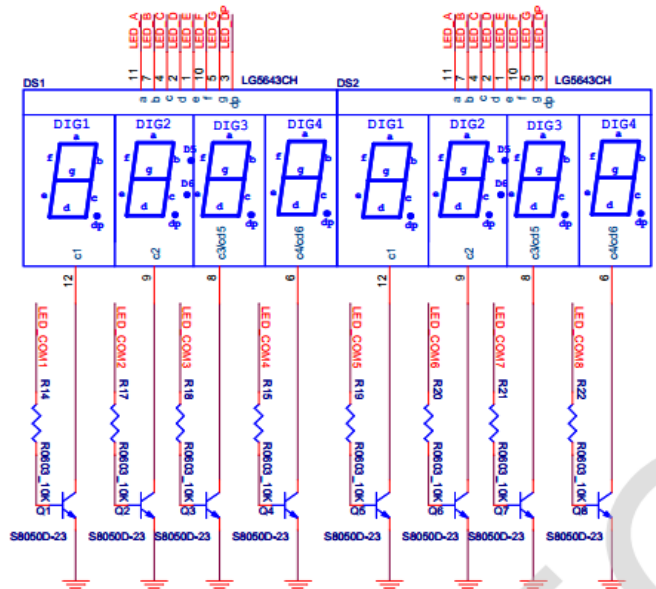
## LEDS



## GPIO EXPAND



## DIGITRON



### ■ 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 configure 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<sup>2</sup>C:**

- 1) To program on a I<sup>2</sup>C module, first you need to supply system clock to that I<sup>2</sup>C module using `SysCtlPeripheralEnable()`;
- 2) If pins used by the I<sup>2</sup>C module is used as GPIO by default, then you need to configure these pins to the proper function;
- 3) Enable the I<sup>2</sup>C Master function of the I<sup>2</sup>C 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:

1. What is the purpose of line `if (UARTCharsAvail (UART0_BASE) )` in `exp3-2.c`? If it is removed from the code, what would happen?
2. Why do we need to read and clear the interrupt status in the handler of UART0 but not in the handler of the SysTick?
3. Program to send commands like “MAY+01” from the host to the S800 so that the S800 will return the result of the command back to the host. The 3-letter “MAY” represents a month in a year (JAN, FEB ... DEC), “+” (or “-”) represents addition (or subtraction), and 2-letter “01” is the amount (ranging from 00 to 11) to be added to (or subtracted from) the given month. As a result, the S800 returns “JUN” back to the host after receiving “MAY+01”.
4. Program to send commands like “14:12+05:06” from the host to the S800 so that the S800 will return the result of the command back to the host. The “14:12” represents 14 minutes and 12 seconds, “+” (or “-”) represents addition (or subtraction), and “05:06” is the amount (i.e., 5 minutes and 6 seconds) to be added to (or subtracted from) the given time. As a result, the S800 returns “19:18” back to the host after receiving “14:12+05:06”.