Subsystem Design

Connected Diode Matrix

1 Description

The Connected Diode Matrix example demonstrates how to use a matrix format to reduce the number of necessary GPIO pins when using six or more LEDs. This specific example uses nine LEDs and six GPIOs to form and control a 3 × 3 LED matrix. The matrix format creates a grid that uses two GPIOs per LED (or diode). This format is especially useful when creating a sign or display out of LEDs. The GPIO pins for the LED matrix are divided into row and column pins. When the row pins connect the cathodes of the LEDs, as Figure 1-1 shows, the matrix is a common row cathode. Common row anode is when the row pins connect the anodes of the LEDs. Depending on the configuration of the LEDs in the LED matrix, the row and column pins are set to active high or active low. For this subsystem example, the row pins are active low and the column pins are active high. For the LED matrix to function properly, the LEDs in the matrix must be controlled one row at a time. The application code for this example uses a state machine to cycle continuously through the rows to turn the LEDs on and off.

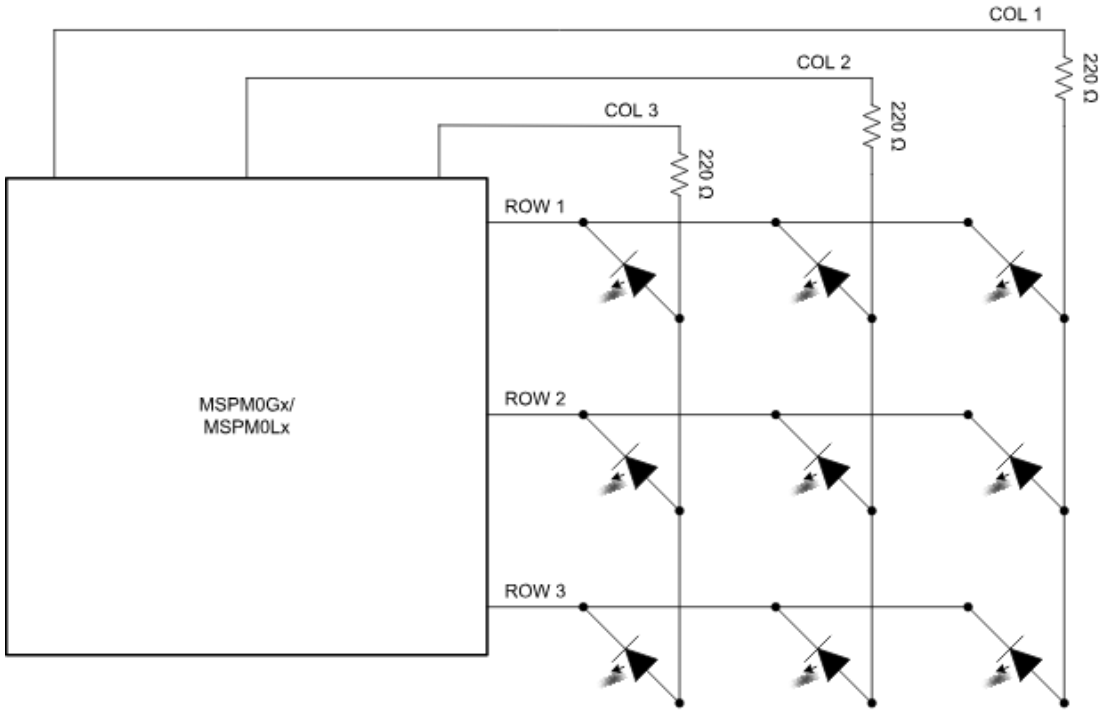

Figure 1-1. Subsystem Functional Block Diagram

2 Required Peripherals

This application requires six GPIO pins and timer interrupts.

Table 2-1. Required Peripherals

Subblock Functionality	Peripheral Use	Notes
GPIO subblock	6 GPIO pins	All pins used for this example are on the same port
Timer	Timer interrupts	Timer interrupts are used to cycle through the rows on the LED matrix

3 Design Steps

1. Determine the number of LEDs used in the matrix as well as the matrix dimensions. The matrix dimensions determine the number of GPIO pins needed.
2. Separate the GPIO pins into row pins and column pins.
3. Configure all row and column pins as outputs.
4. Determine the mask value for the column pins by taking the bit-wise OR of all the column pin GPIO values.
5. Create the memory table and the memory table update function.
6. Create an enumeration table for the row update state machine to cycle between rows.
7. Configure timer interrupts and write application code for the row update state machine and to increment the LED state.
8. Write application code to set the display period and to update the memory table with new column pin values as the display changes.

4 Design Considerations

1. **Number of LEDs and matrix dimensions:** The matrix dimension determines the number of GPIO pins needed to run the matrix. For example, a 16 LED matrix can use 8 pins in a 4 × 4 matrix or 10 pins in a 2 × 8 matrix.
2. **LED configuration:** The active states of the row and column pins is dependent on whether the matrix is in common row cathode or common row anode.
3. **Column pin values:** Column pin values are set in a memory table. The exact values are determined by which pins are selected and the respective column mask. For ease of setup, selecting pins that are in numerical order with no gaps is the easiest.
4. **Column and row pin connections:** When connecting pins to the LED matrix, application programming is easiest if the row pins start from the topmost row (moving down) and the column pins start from the right-most column (moving left).
5. **Timer interrupts:** The speed of the interrupts affects the display period and how long each row of LEDs is on per the state machine cycle. This specific example interrupts every 5ms, preventing the human eye from noticing any flickering.
6. **Updating the memory table:** The specific method of updating the memory table depends on the application. This example increments a counter (otherwise known as the display period) up to a specified value. When the counter reaches that value, the memory table is updated to set a new display.

5 Software Flow Chart

Figure 5-1 shows the software flow chart for this subsystem example and explains the timer interrupt routine and state machine used to control the LED matrix.

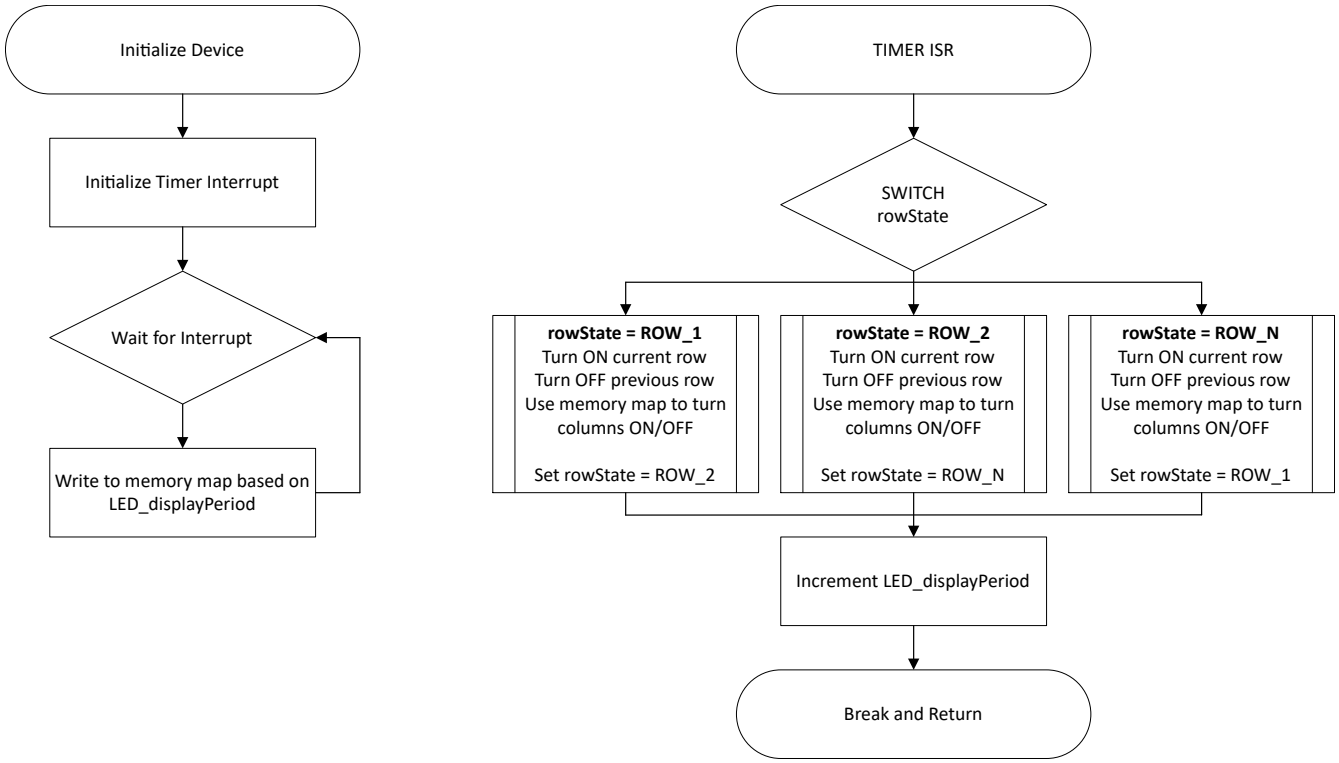

Figure 5-1. Application Software Flow Chart

6 Application Code

This application makes use of the TI System Configuration tool (SysConfig) graphical interface to generate the configuration code for the device peripherals. Using a graphical interface to configure the device peripherals streamlines the application prototyping process.

There are a few key variables that this example uses: the number of rows, the column mask value, the display period duration, and a counter to track the number of interrupts. The number of rows is a defined value that is used to build the memory table array. The column mask is equivalent to the bitwise OR of the GPIO values of all of the column pins used. The column mask is used with the memory table to determine which column pins need to be on or off per row at a given time. The display period variable is multiplied by the duration of time per timer interrupt to determine the amount of time that a single memory table write is used. For this example, the display period value is defined as 100 which equates to a display period time of half a second. The counter, or gLedState, is used to track the number of interrupts in relation to the display period value. This makes sure that the memory table is written to every display period.

```
#define NUMBER_OF_ROWS 3
#define COL_MASK 0x38
#define LED_DISPLAY_PERIOD 100 /* timer period = 5 ms, so display period = 500 ms */
volatile uint32_t gLedState = 0;
void LED_updateTable(uint8_t rowNumber, uint8_t LEDs);
```

The next snippet of code shows the enumeration table as well as the timer Interrupt Request (IRQ). The enumeration table defines the row states that the rowState switch cycles through in the timer IRQ. For each rowState (or row pin), the current row is turned on, the previous row is turned off, and the columns are set by comparing the column mask value with the memory table value. The next rowState is then set. This example cycles sequentially from row one to row N and back to one. Before leaving the timer IRQ, gLedState is incremented to track the number of interrupts for each display period.

```
typedef enum {
    ROW_1,
    ROW_2,
    ROW_3
}rowNumber;

rowNumber rowState = ROW_1;

void LED_STATE_INST_IRQHandler(void) {
    switch (DL_TimerG_getPendingInterrupt(LED_STATE_INST)){
        case DL_TIMER_IIDX_ZERO:
            /* State machine to auto cycle from row 1 to row N and repeat */
            switch (rowState){
                case ROW_1:
                    /* Turn on ROW_1, Turn off ROW_3 */
                    DL_GPIO_clearPins(ROW_PORT, ROW_ROW_1_PIN);
                    DL_GPIO_setPins(ROW_PORT, ROW_ROW_3_PIN);

                    /* Set COLUMN values */
                    DL_GPIO_writePinsVal(COLUMN_PORT, COL_MASK, gLedMemoryTable[0]);
                    rowState = ROW_2;
                    break;
                case ROW_2:
                    /* Turn on ROW_2, Turn off ROW_1 */
                    DL_GPIO_clearPins(ROW_PORT, ROW_ROW_2_PIN);
                    DL_GPIO_setPins(ROW_PORT, ROW_ROW_1_PIN);

                    /* Set COLUMN values */
                    DL_GPIO_writePinsVal(COLUMN_PORT, COL_MASK, gLedMemoryTable[1]);
                    rowState = ROW_3;
                    break;
                case ROW_3:
                    /* Turn on ROW_3, Turn off ROW_2 */
                    DL_GPIO_clearPins(ROW_PORT, ROW_ROW_3_PIN);
                    DL_GPIO_setPins(ROW_PORT, ROW_ROW_2_PIN);

                    /* Set COLUMN values */
                    DL_GPIO_writePinsVal(COLUMN_PORT, COL_MASK, gLedMemoryTable[2]);
                    rowState = ROW_1;
                    break;
            }

            /* Increment LED_STATE */
            gLedState++;

            break;
        default:
            break;
    }
}
```

In the main code, all that is done is to write to the memory table every display period. This repeats indefinitely. This particular code uses binary to make determining which LED is on easier as the layout of 1s and 0s mimics the matrix layout. The binary value is 1 if an LED is on and 0 if an LED is off.

```
while(1){
    __WFI();
    /* Flash TI on repeat in half second increments */
    if (gLedState == LED_DISPLAY_PERIOD){ /* Display "T" for one display period */
        LED_updateTable(1, 0b111);
        LED_updateTable(2, 0b010);
        LED_updateTable(3, 0b010);
    } else if (gLedState == LED_DISPLAY_PERIOD*2){ /* Blank for one display period */
        LED_updateTable(1, 0b000);
        LED_updateTable(2, 0b000);
        LED_updateTable(3, 0b000);
    } else if (gLedState == LED_DISPLAY_PERIOD*3){ /* Display "I" for one display period */
        LED_updateTable(1, 0b111);
        LED_updateTable(2, 0b010);
        LED_updateTable(3, 0b111);
    } else if (gLedState == LED_DISPLAY_PERIOD*4){ /* Blank for one display period */
        LED_updateTable(1, 0b000);
        LED_updateTable(2, 0b000);
        LED_updateTable(3, 0b000);
    } else if (gLedState > LED_DISPLAY_PERIOD*4){ /* Reset gLedState and start over */
        gLedState = 0;
    }
}
```

7 Hardware Design

This specific subsystem example requires nine LEDs, three resistors, and at least six wires. To setup the matrix, arrange the LEDs in 3 × 3 rows. Connect the cathodes of each row of LEDs together. Then, connect the anodes of each column of LEDs together. Connect a 220Ω resistor to each column line. From there, connect the row lines and column lines to the correct device pins based on the device configuration. See Figure 1-1 for connection guidelines.

8 Results

Figure 8-1 showcases the intended results of the "T" display period for this application. The top half of the figure shows the state of each row as the application state machine cycles through each row per interrupt. The bottom half of the figure shows what the composite image over a full cycle is. This is how the matrix appears to the human eye.

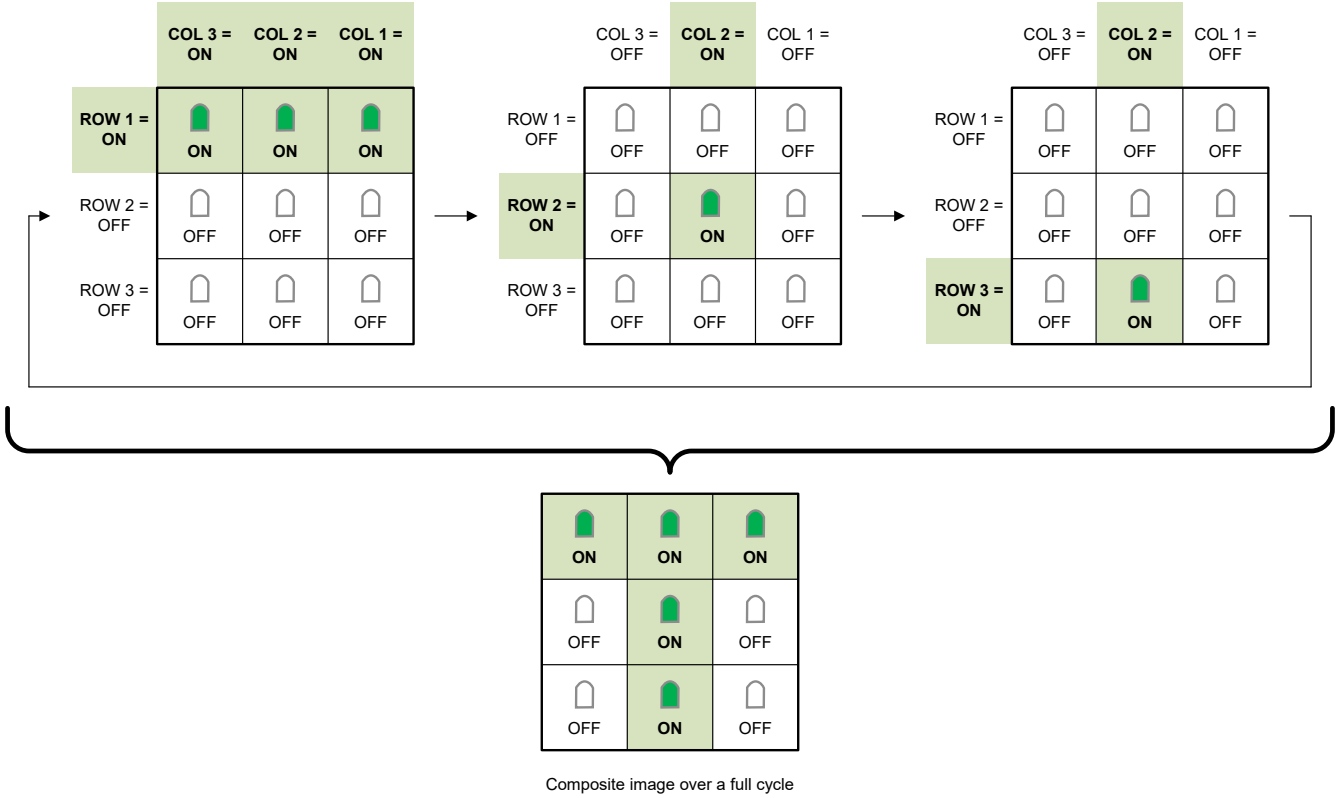

Figure 8-1. Results

9 Additional Resources

- Texas Instruments, Download the MSPM0 SDK
- Texas Instruments, Learn more about SysConfig
- Texas Instruments, MSPM0L LaunchPad™
- Texas Instruments, MSPM0G LaunchPad™
- Texas Instruments, MSPM0 Academy

10 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (September 2024) to Revision A (August 2025)	Page
• Removed Compatible Devices section	2

11 E2E

See TI's E2E™ support forums to view discussions and post new threads to get technical support for utilizing MSPM0 devices in designs.

12 Trademarks

LaunchPad™ and E2E™ are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265