Programming with the Intel Galileo

External Registers

Why use an External Register

Using an external register, such as a shift register, can increase the number of microcontroller output pins available for other purposes. For example, suppose a project requires 8-bits to be displayed on 8 LEDs. The most straightforward approach would be to output the 8-bits using 8 microcontroller pins. Instead, by using a shift register, only 5 pins are needed, thus freeing 5 pins for other uses.

What the Shift Register Does

The shift register used in this lab contains two internal 8-bit registers named the “Shift” register and the “Storage” register. The “Shift” register is used to serially receive and store up to eight bits of data. The data stored in this register cannot be displayed on the output pins (Qa-Qh). However, the data stored in the “Shift” register can be copied, in parallel, to the “Storage” register at any time. From the “Storage” register, the data can be displayed on the output pins if output is enabled. Shifting data into the “Shift” register does not affect the contents of the “Storage” register.

A diagram of the shift register used in this lab is given below. The datasheet can be found in the GDrive folder under 'datasheets'.

---

**Shift Register Diagram**

Model #: SN74HC595N

- Qb
- Qc
- Qd
- Qd
- Qe
- Qf
- Qh
- Ground
- Vcc
- Qa
- Serial In (SER)
- Output Enable (OE)
- Storage Reg Clock (RCLK)
- Shift Reg Clock (SRCLK)
- Shift Reg Clear (SRCLR)
- Qh'
How to Operate the Shift Register

1) Transmitting Data

- On the shift register, four pins are necessary for serially transmitting, and storing, up to a byte of data: SRCLR, SER, SRCLK, and RCLK. These can be connected to any four pins on the microcontroller -- we will outline below the preferred pins for this lab.
- When ready to begin transmitting, set SRCLR high and RCLK low.
- The following three steps should be repeated for every bit to be transferred.
  - 1) Set SRCLK low.
  - 2) Set SER equal to the bit value to be transmitted (0 or 1).
  - 3) After SER has been set, then set SRCLK high. The rising edge of SRCLK will shift in the contents of SER to the first spot of the "Shift" register and the remaining contents of the "Shift" register will be shifted over one spot. The bit stored in the most significant bit of of the "Shift" register will be 'bumped off' and lost.
- After all of the data has been transmitted, set RCLK high. This copies the contents of the "Shift" register to the "Storage" register. The copy takes place on the rising edge of RCLK.

NOTE: To conserve a pin, RCLK and SRCLK can share the same clock line. Keep in mind that if they do, the contents of the “Shift” register will be a single clock cycle ahead of the contents of the “Storage” register. For example, shifting from right to left, if the “Shift” registers contains the value: 0000 1111, then the “Storage” register will contain: 0000 0111.

- After copying the contents of the “Shift” register to the “Storage” register, drop SRCLR low to clear the “Shift” register and cease transmitting data to the shift register. If SRCLR is dropped low before setting RCLK high, the transmitted data will be cleared before it can be stored.
PINS 1-7, 15  |  Q0 " Q7  |  Output Pins  
PIN 8       |  GND    |  Ground, Vss  
PIN 9       |  Q7"    |  Serial Out   
PIN 10      |  MR     |  Master Reclear, active low 
PIN 11      |  SH_CP  |  Shift register clock pin (Clock Pin) 
PIN 12      |  ST_CP  |  Storage register clock pin (latch pin) 
PIN 13      |  OE     |  Output enable, active low 
PIN 14      |  DS     |  Serial data input (Data Pin) 
PIN 16      |  Vcc    |  Positive supply voltage 

//First Shift Register  
#define LATCHPINONE 10  
#define CLOCKPINONE 12  
#define DATAPINONE 11  

void setup(){  
  pinMode(LATCHPINONE, OUTPUT);  
  pinMode(DATAPINONE, OUTPUT);  
  pinMode(CLOCKPINONE, OUTPUT);  
}  

Operating a Shift Register on Arduino and the Intel Galileo

It is recommended to use certain pins on the Intel Galileo because some pins operate faster. Pins 10, 12, and 11 work well with the shift register like the code sample above. Make sure to set the pinMode as OUTPUT for these as well.

void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val)

Arduino provides a function shiftOut for sending data to shift registers.

How to use it:

| void transmitData(unsigned char data)| {  
|   digitalWrite(LATCHPINONE, LOW);  
|   shiftOut(DATAPINONE, CLOCKPINONE, MSBFIRST, data);  
|   digitalWrite(LATCHPINONE, HIGH);  
|}  

Video Demonstration: https://youtu.be/Q3cESBmWLO0

Part 1: Incrementing/Decrementing a Value on a Single Shift Register

Design a system where a 'char' variable can be incremented or decremented based on specific button presses. The value of the variable is then transmitted to the shift register and displayed on a bank of eight LEDs.

Criteria:
- Two buttons are required. One button increments the variable. The second button decrements the variable.
- The variable cannot be incremented past 0xFF or decremented below 0x00.
- If both buttons are pressed simultaneously, the variable is reset to 0x00.
- The variable is transmitted, and displayed, on the shift register whenever the variable changes value.

Part 2: Festive Light Display

Design a system where one of three festive light displays is displayed on the shift register’s LED bank. The choice of festive light displays is determined by button presses (See video link for examples of some festive light displays).

Criteria:
- Two buttons are used to cycle through the three available festive light displays
- One button cycles up, the other button cycles down.
- Pressing both buttons together toggles the system on or off
- Choose a default display for the system to start with. Whenever the system is turned on, the display shown on the LEDs should start with the default display.
Part 3: Two Festive Light Displays

Expand upon part 2 of the lab by adding another shift register and two buttons. Design a system where two different festive light displays can be displayed on two separate shift registers simultaneously. The selected festive light display for each shift register, is determined by the two buttons designated to that shift register.

Criteria:

- Two buttons control the festive light display shown on shift register 1. The other two buttons control the festive light display shown on shift register 2.
- For each shift register, if their designated buttons are pressed simultaneously, the LED display on that shift register is toggled on or off.

Part 4 (Challenge) : Daisy Chaining Shift Registers

When shift registers are "Daisy Chained" together, this means that the shift registers are connected in series where the output of one shift register is connected to the input of another shift register. "Daisy Chaining" shift registers together allows for eight additional outputs for the cost of two microcontroller pins.

"Daisy Chain" two shift registers which displays a single illuminated LED that shifts back and forth across all 16 LEDs, bouncing the opposite direction when the illuminated LED hits the far left or far right LED.

Criteria:

- "Daisy Chain" Sreg1 with Sreg2 by connecting Qh of Sreg1 to SER of Sreg2.
- For each register, Sreg1 and Sreg2, share the local RCLK and SRCLK clock line, but each register has it’s own independent clock line.
Hints:

- Calling shiftOut twice would reach one shift register at a time.
Part 5 (Challenge): The Jumping Game

Using two “Daisy Chained” shift registers, design the following game. A player controller illuminated LED needs to avoid a collision with a game controlled (enemy) illuminated LED. The player can move their LED left or right, and can jump left or right. A collision can be avoided by jumping over the enemy LED.

Criteria:

● Three buttons control the player LED: Left, Right, and Jump
● Pressing Left or Right on their own will shift the player LED by one bit in the desired direction
● A jump occurs when the Jump button is pressed simultaneously with Left or Right button. The direction of the jump is determined by what directional button is pressed with the Jump button.
● A jump should shift by 4 bits.
● A jump should not shift past an extreme. For example, jumping right from 0x0004, should result in 0x0001 instead of 0x0000;
● The enemy LED shifts by one bit in one direction until it hits an extreme, then changes directions. For example, if the enemy is shifting left and equals 0x8000, the enemy should now change direction and begin shifting right until it equals 0x0001.
● The Enemy LED moves at a constant rate