```
#include <avr/io.h>

#define F_CPU 8000000UL  // 8 MHz clock frequency
#define HX711_DT_PIN PD2  // Data pin connected to digital pin PD2
#define HX711_SCK_PIN PB1 // Clock pin connected to digital pin PB1


// Function to send data to MAX7221 display driver
void send_to_MAX7221(unsigned char command, unsigned char data) {
	PORTB &= ~0b00000100;  // Set LOAD/CS low

	SPDR = command;  // Send command
	while(!(SPSR & (1<<SPIF)));  // Wait for transmission to complete

	SPDR = data;  // Send data
	while(!(SPSR & (1<<SPIF)));  // Wait for transmission to complete

	PORTB |= 0b00000100;  // Set LOAD/CS high
}

// Function to display numbers using MAX7221
void display_number(uint32_t currentweight) {
	uint32_t lowweight = 8360000;
	uint32_t maxweight = 8565000;

	if (currentweight < lowweight) {
		currentweight = lowweight;
	} else if (currentweight > maxweight) {
		currentweight = maxweight;
	}

	int percent = (currentweight - lowweight) / ((maxweight-lowweight) / 100);

	if (percent > 99) {
		percent = 99;
	}

	unsigned char tens = (unsigned char)(percent / 10);
	unsigned char units = (unsigned char)(percent % 10);

	send_to_MAX7221(0x01, tens);
	send_to_MAX7221(0x02, units);
}
```

```
// Function to read 24-bit data from the HX711
uint32_t HX711_read() {
	uint32_t count = 0;
	while (PIND & (1 << HX711_DT_PIN)); // Wait for DT to go low

	for (uint8_t i = 0; i < 24; i++) {
		PORTB |= (1 << HX711_SCK_PIN);  // Set SCK high
		count = count << 1;		// Shift the bits to the left
		PORTB &= ~(1 << HX711_SCK_PIN); // Set SCK low
		if (PIND & (1 << HX711_DT_PIN)) { // If DT is high
			count++;
		}
	}
	// Set the gain for the next reading (default is 128)
	PORTB |= (1 << HX711_SCK_PIN);  // Set SCK high
	count ^= 0x800000;		// Convert to signed value if necessary
	PORTB &= ~(1 << HX711_SCK_PIN); // Set SCK low

	return count;
}


void wait(volatile int numMilliseconds) {
	for (volatile int x = 0; x < numMilliseconds; x++) {
		for (volatile int y = 0; y < 175; y++) {
			// Brute force wait
		}
	}
}

int main(void) {

	DDRD &= ~(1 << HX711_DT_PIN);  // Set DT pin as input
	DDRB |= (1 << HX711_SCK_PIN);  // Set SCK pin as output
	PORTB &= ~(1 << HX711_SCK_PIN); // Initialize SCK pin to low

	DDRC = 0b00001011;  // Set LED and H-bridge pin as output
	PORTC |= (1 << 0);
	PORTC |= (1 << 1);
	PORTC |= (1 << 5);

	DDRB |= (1 << PB3) | (1 << PB5) | (1 << PB2);
	SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);  // SPI setup
	send_to_MAX7221(0x0B, 0x01);   // Display digits 0 and 1
```

```
send_to_MAX7221(0x09, 0xFF);   // Use code B decoding for all digits
send_to_MAX7221(0x0C, 0x01);   // Exit shutdown mode
send_to_MAX7221(0x0A, 0x0F);   // Set brightness (0x00 - 0x0F)
send_to_MAX7221(0x0F, 0x00);   // Disable display test

int openValve = 0;
int valveAutoClosePause = 0;
uint32_t weight = 0;
uint32_t openValveWeight = 8500000;

while (1) {
	weight = HX711_read();  // Read the weight data from HX711
	display_number(weight);

	valveAutoClosePause = 0;
	if (weight < openValveWeight) { // Set threshold based on your calibration
		openValve = 1;
		valveAutoClosePause = 800;
	} else if (!(PINC & (1 << 5))) {
		wait(150);
		if (!(PINC & (1 << 5))) {
			openValve = 1;
		} else {
			openValve = 0;
		}
	} else {
		openValve = 0;
	}

	if (openValve) {
		PORTC &= ~(1 << 1);
		PORTC |= (1 << 3);

		if (valveAutoClosePause > 0) {
			wait(valveAutoClosePause);
			PORTC &= ~(1 << 3);

			PORTC &= ~(1 << 0);
			wait(3000);  // Wait for the scale to stabilize
			PORTC |= (1 << 0);
		}
		} else {
		valveAutoClosePause = 0;
		PORTC &= ~(1 << 3);
```

```
                PORTC |= (1 << 1);
            }

            // Add logic to display numbers as needed
        }
        return 0;
}
```