Concordia University
Department of Computer Science and Software Engineering

SOEN422 Fall 2021

—

Lab 3 (10%)

—

Building HAL and BSL layers for the Arduino Nano (Part A)

Purpose

This lab aims to build HAL and BSL layers for the Arduino Nano (Part A). It will have six tasks.

Task 1 [1%]: Write functions to manage hardware interrupts

Build the following portable functions to manage hardware interrupts for the HAL layer based on the following header file **hal_Interrupt.h**:

```
// hal_Interrupt.h - HAL Interrupt Management

#ifndef __hal_Interrupt_h
#define __hal_Interrupt_h

#include <stdint.h>

void     hal_Interrupt_Disable(void);
void     hal_Interrupt_Enable(void);
uint32_t hal_Interrupt_SaveAndDisable(void);
void     hal_Interrupt_Restore(uint32_t flags);
```

Both `SaveAndDisable` and `Restore` functions need to access the `SREG` status register. See lecture notes Section 5.3.4 and ATmega368 datasheet for more details.

Task 2 [0.5%]: Write macros and functions to manipulate bits for the HAL layer

Build portable macros to manipulate bits within a utility sub-layer for the HAL layer based on the following header file **hal_util_BitMacros.h** to fulfill:

```
// hal_util_BitMacros.h - HAL utility macros to manipulate bits

#ifndef __hal_util_BitMacros_h
#define __hal_util_BitMacros_h

// All macros return the resulting value from the specified value.

#define hal_util_Bit_Read(value,bit)             ...
#define hal_util_Bit_Set(value,bit)              ...
#define hal_util_Bit_Clear(value,bit)            ...
#define hal_util_Bit_Toggle(value,bit)           ...
#define hal_util_Bit_Write(value,bit,bitvalue)   ...

#endif // __hal_util_BitMacros_h
```

Build portable functions to manipulate bits within a utility sub-layer for the HAL layer based on the following header file **hal_util_Bit.h**:

```
// hal_util_Bit.h - HAL utility functions to manipulate bits

#ifndef __hal_util_Bit_h
#define __hal_util_Bit_h

#include <stdint.h>
#include <stdbool.h>

// All functions return the resulting value from the specified value.

uint8_t hal_util_Bit_Read  (uint8_t value, uint8_t bit);
uint8_t hal_util_Bit_Set   (uint8_t value, uint8_t bit);
uint8_t hal_util_Bit_Clear (uint8_t value, uint8_t bit);
uint8_t hal_util_Bit_Toggle(uint8_t value, uint8_t bit);
uint8_t hal_util_Bit_Write (uint8_t value, uint8_t bit, bool bitvalue);

#endif // __hal_util_Bit_h
```

Task 3 [2%]: Write a bargraph ADT software component to manage LEDs

Build a Bargraph ADT component that manages from one up to four LEDs (HAL interface) connected to digital pins 8 through to 11 (D8..D11) on the Arduino Nano (BSL interface). For this task, interconnect the MV57164 bargraph display is a 10-element bar graph array of 10 red LEDs[1] and one of the two Bourns resistor networks[2] on your breadboard with the digital pins D8 to D11 inclusive[3].

The Bargraph ADT should provide the following public interface:

```
// hal_Bargraph.h

#ifndef __hal_Bargraph_h
#define __hal_Bargraph_h

#include <stdint.h>

        struct hal_BargraphDesc;                // Forward reference
typedef struct hal_BargraphDesc* hal_Bargraph;  // Opaque type

hal_Bargraph hal_Bargraph_New(uint8_t numberOfLeds);
void         hal_Bargraph_Set(Bargraph _this, uint8_t ledID);
void         hal_Bargraph_Clear(Bargraph _this, uint8_t ledID);
uint8_t      hal_Bargraph_GetNumberOfLeds(Bargraph _this);
void         hal_Bargraph_Delete (hal_Bargraph _this);

#endif // __hal_Bargraph_h
```

The `numberOfLeds` and `ledID` must be validated in your implementation. The `New` returns a null pointer if its parameter is not in the 1..4 in the BSL layer since you are supposed to connect 4-LEDs in the MV57164 bargraph display. `Set` and `Clear` functions will do nothing if the `ledID` parameter is not in the validated range (between `0..numberOfLeds-1`). And finally, the `GetNumberOfLeds` function returns a number which is the validated number of LEDs at creation.

Make a test program unit on target showing the proper behavior of your Bargraph ADT.

[1] See datasheet: MV57164 Fairchild, Jan 2005.
[2] Bourns 10X-I-681LF or Bourns 10X-I-581LF.
[3] D8 is PB0 (Led 1), D9 is PB1 (Led 2), D10 is PB2 (Led 3), and D11 is PB3 (Led 4).

Task 4 [1.5%]: Make a port of an existing version of Memory Manager

Make a port of an existing version of Memory Manager[4] as a portable Memory Manager sub-layer for the HAL layer (BSL for ATmega368). Make sure to replace all `typedef` used in this prior version (see the `cddef.h` header file) with the following standard headers: `stdint.h` and `stdbool.h`. You must configure the memory manager with only one partition of 16 blocks where each block has 64 bytes representing 1024 bytes.

The HAL memory manager should provide the following public interface:

```
// hal_MM.h - HAL Memory Manager (MM) for segments of blocks.

#ifndef __hal_MM_h
#define __hal_MM_h

#include <stdint.h>

void  hal_MM_Init(void);
void* hal_MM_GetSegment(uint16_t nBytes);
void  hal_MM_FreeSegment(void* segmentBaseAddress);

#if defined(MM_DefragOn)
void  hal_MM_Defrag(void);
#endif

#endif __hal_MM_h
```

The BSL memory manager should provide the following internal interface:

```
// bsl_MM.h - BSL Memory Manager (MM) to get the number of free blocks.

#ifndef __bsl_MM_h
#define __bsl_MM_h

#include <stdint.h>

uint8_t bsl_MM_GetNumberOfFreeBlocks(void);

#endif __bsl_MM_h
```

This internal function `GetNumberOfFreeBlocks` must be implemented in a separate file `bsl_MM.c` and will provide an internal bsl_MM.h public interface to get access to the internal `bytemap` array to traverse it and calculate the number of free blocks visualize the current memory partition.

Make a test program unit on the target (Arduino Nano) for your demo for testing these two interfaces: `hal_MM.h` and `bsl_MM.h`.

This HAL MM component and these two interfaces will be reused by **Task 6**.

[4] C source files given on Moodle: `Lab3_MMsrcFiles.zip`

Task 5 [3%]: Write a pedestrian crossing signals application

Write a pedestrian crossing signals application by reusing and configuring your Bargraph ADT component for 3-LEDs. Digital pin 12 (D12) will emulate a button to cross the street with a jumper wire.

The "Walk" Effect
When you contact pin D12 (PB4) with a jumper wire to GND[5], it should turn on the OnBoard LED, start the "Walk" effect on the bargraph to appear moving along the line, then restart to the start like a rotating effect for 10 seconds.

The "Flashing" Effect — Don't Walk Effect
The walk effect will change to the "don't walk" or "wait" effect after 10 seconds. The don't walk/wait effect is flashing the 3-LEDs rapidly for 5 seconds and after that turning all LEDs off.

Task 6 [2%]: Write a memory manager viewer application

Write a memory manager viewer application by reusing and configuring your Bargraph ADT component for 4-LEDs. The 4-LEDs will proportionally represent the 1024 bytes contained in the Memory manager.

The first LED will light when 25% of the memory is used; the first two LEDs will light when 50% is used, and so on. Make a small bare-metal application in C using `hal_MM.h` public interface to allocate and deallocate blocks of memory and the `bsl_MM.h` internal interface to control the bargraph to visualize the current memory usage in percentage (0% = 0 LED on, 25% = 1 LED on, 50% = 2 LEDs on, 75% = 3 LEDs on, 100% = all LEDs on).

Individual Lab Report Submission

Each student is expected to create a `.zip` file that he will upload to Moodle for grading. The following is expected in the archive:

- A folder containing your source code
- A report in **.PDF** format, no other formats will be accepted

The individual lab report is expected to contain the following for each task:

1. Listing of all your code (header files, implementations files, and tests files).
   Your code must respect the naming convention (mentioned in Lecture 4) and have comments explaining the logic behind it. The code for each task should be in separate source files.
2. Discussion and conclusion.

Create a `.zip` file with the following signature:
Soen422_LabSection_Lab3_Lastname_StudentID.zip

Due 11:59 PM - Monday, October 18, 2021

[5]Don't forget to set PB4 as an input with pull-up.