SPI Practice Problems

1. Create a C function, *void spie_init(void)*, to configure the appropriate SPIE module such that [1] a serial clock frequency 32 kHz is utilized (assume that the system clock frequency is **64 kHz**), [2] data transmitted will be arranged in a most-significant-bit-first ordering, [3] every transmission of a bit will occur upon on a rising edge of the serial clock, [4] the default value of the serial clock, when not in use, will be zero (low), and [5] the module is enabled. Be sure to review Figure 22-2 within *doc8331*. Additionally, within the function, appropriately configure the data direction and output values for all relevant I/O port signals related to the SPIE module. Assume that the SPI system will be inactive unless new SCI data is received.

C Code	Comments/More C Code

SPI Practice Problems

2. Solve each of the following short problems.

   a) Is the clock frequency for a SPI receiver faster than that of the corresponding transmitter? If so, why? If not, why not?

   b) Why are there four SPI transfer modes (see Figure 22-2 in *doc8331*) provided by the *ATxmega128A1U* (and most other microcontrollers)?

   c) Describe the main advantage of using synchronous serial communication (i.e., communication via a protocol such as SPI) instead of asynchronous serial communication (SCI) for bidirectional communication assuming the same bit rates.

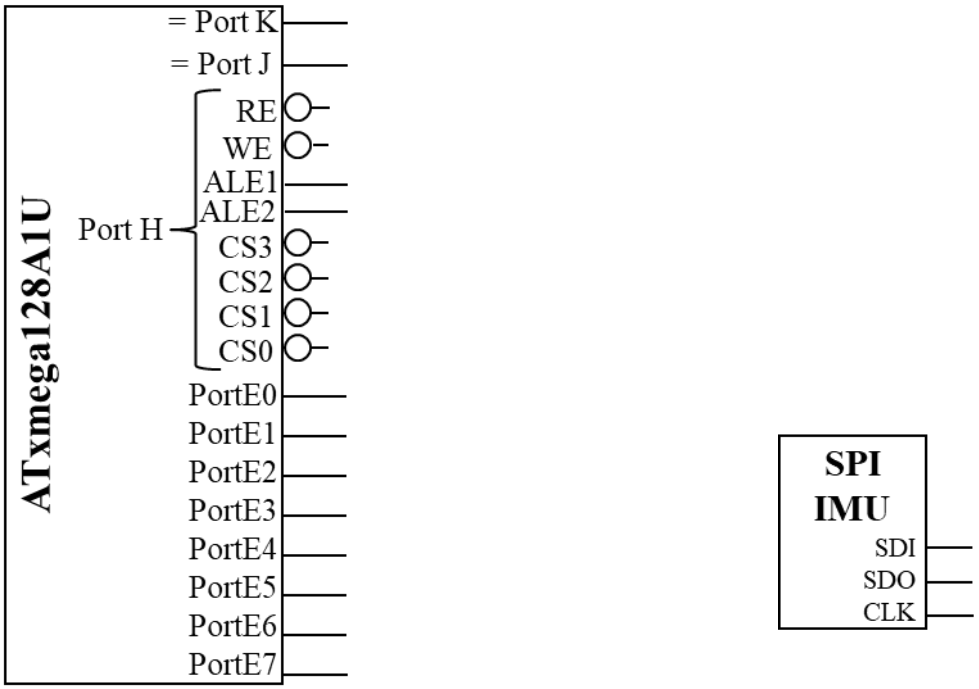
   d) Describe the main advantage of using asynchronous serial communication (SCI) instead of synchronous serial communication (i.e., communication via SPI) for bidirectional communication assuming the same bit rates.

3. In this problem, you will design a simple subsystem of some robot car and write a C program to utilize it. The subsystem will contain [1] an *ATxmega128A1U* microcontroller (assumed to have a system clock frequency of 16 MHz), and [2] an inertial measurement unit (IMU) much simpler than the *LSM6DSL*, consisting solely of an accelerometer, that performs communication via an internal SPI system. A relevant block diagram for each component is given below. (Within the IMU block diagram, *SDI* and *SDO* represent *serial data input* and *serial data output*, respectively, and *CLK* represents the relevant serial *clock*.)

   Overall, it is intended that the microcontroller must request three consecutive bytes of data from the IMU roughly every ten milliseconds, where some TC0 module and its appropriate interrupt should be configured to keep track of ten millisecond intervals.

   The three consecutive bytes to be read from the IMU will represent signed data for the X, Y, and Z axes of the accelerometer, respectively (i.e., the first byte will represent data for the X-axis of the accelerometer, the second byte for the Y-axis of the accelerometer, and the third byte for the Z-axis of the accelerometer). Data can be immediately requested from the IMU by writing any value, i.e., there is no need to first specify an address or anything similar, however [1] the serial clock frequency must be **at most** 3 MHz (but at least 1MHz), [2] data should be set up on a rising edge and sampled on a falling edge, with the least-significant bit being transmitted first, and [3] the serial clock should be high when no data is being transmitted.

   a) Assuming that all components are properly powered, complete the circuit diagram below for the relevant hardware expansion, **using only the appropriate signals and components. Additionally, whenever appropriate, use legible labels to identify physical connections instead of drawing lines.**



3. b) Following the relevant previously stated assumptions (including the assumption that the system clock frequency is 16 MHz), create a C function, *void spi_init(void)*, to initialize the appropriate SPIE module to be able to communicate with the relevant IMU.

C Code	Comments/More C Code

3. c) Assume that the relevant SPI module is already configured, and assume that both the array *int8_t accel_data[3]* and the function *void imu_report(int8_t *a_data)* are already defined for you. Below, create a C function, *void imu_read(void)*, to [1] retrieve the relevant three bytes of data from the IMU, storing X, Y, and Z accelerometer data into *accel_data* array locations 0, 1, and 2, respectively, and then to [2] pass the *accel_data* array to the *imu_report* function, i.e., *imu_report(accel_data)*. You may *not* utilize any previously written SPI routine.

C Code	**Comments/More C Code**

3. d) Assuming that the system clock frequency is 16 MHz and that a TC0 prescaler of sixty-four is necessary, create a C function, *void timer_init(void)*, to initialize some TC0 module such that an overflow interrupt is generated roughly every ten milliseconds. Do not configure the PMIC system, nor set the global interrupt enable bit in this function.

C Code	Comments/More C Code

e) Design an interrupt service routine for the relevant TC0 module (using the relevant C macro function) to simply call the previously written function *void imu_read (void)*.

C Code	Comments/More C Code

3. f) Assuming that the previously provided variables and functions, as well as the previously required functions, can be referenced via the header file *"practice_spi.h",* write the remainder of a complete C program (i.e., any relevant compiler directives, a main routine, etc.) to simply initialize the relevant systems and enter an infinite loop. Any functions previously written for this problem should be utilized whenever appropriate, and previously written interrupt service routines do not need to be rewritten. Additionally, assume that the system clock frequency is already configured to be 16 MHz.

C Code	**Comments/More C Code**