Assignment notes general

1 Introduction

* The term project aim for creating a computer controlled electromechanical ping pong game.
* The main challenge will be to assemble the hardware components and develop software for the microcontrollers
  1. system overview
* the system is divided into two main parts
  + node 1
    - breadboard and the USB Multifunction card
  + node 2
    - Arduino Mega560 developments board, and a dedicated I/O-card
* In the first exercises, only Node 1 will be used. Node 2 will be added later on when including the game board and the CAN bus.
  1. Practical information
* Groups
* Lectures
* Schedule
* Approval exercises
* Lab
* Documentation and datasheets
  + All datasheets will be published on Blackboard
  + Atmel Studio’s help system also contains a lot of valuable information. For information about the various development tools and debugging devices.
* Evaluation
  + Groups must present their project and demonstrate, an oral presentation is also required to the course teacher.
  + No written reports are required
  1. How to work with the project
* The project is comprehensive and time consuming. Be careful and thorough when implementing each part
* To ensure modularization and reusability, organize your code as compact drivers with logical interfaces.
* There might be a lot of delays due to faulty components, hard-to-find bugs or difficult tasks

1. General background information
   1. Hardware and electronics
   2. Breadboard

* Node one consist of a breadboard where you are going to place ICs and connect them by wires. You are strongly recommended to follow the direction for IC placement in figure 5. Also make sure to place the components in the correct direction
  1. Noise, grounding and decoupling
* Electrical noise is an inherent problem with all breadboards. Proper decoupling of all ICs and ensuring good ground connection, especially to the crystal and microcontroller is extremely important.
* All ground and voltage supply should originate from the same physical point.
* Decoupling
  + The supply lines in the circuits are not ideal as they all introduce a certain amounts of impedance between the power source and the IC. The power consumption of the ICs are not constant as they withdraw a highly variable amount of current while operating. The power source can only accommodate changing demands in the current supply up to a certain frequency. This combination will result in noise like voltage drops occurring at the voltage supply pins of the IC.
  + Solution: decoupling capacitor.
    - The capacitor is connected between ground and the voltage supply pins of the IC.
    - It will then operate as a fast local current source that will counteract voltage drops during transient changes in IC power demand.
* Star-point grounding
  + Connect all ground return wires to a common point close to the ground in a star-like connection
* Power drain
  + Isolating the solenoid to a separate voltage supply.
  1. LEDS
* Can be a valuable debugging tool for instance when used to display the logical value of a signal.
* They serve as indicator that e.g. power supply is enabled, system state is initialized
* You can have active low or active high configuration

2.1.4 Pull-up and pull-down resitors

* Read later
  1. The Atmel AVR microcontrollers
* The Atmel AVR microcontrollers used in this project are based on an 8-bit modified Harward architecture (separate memory for program and data).
* RIISC architecture implies that almost every instruction can be carried out in just one clock cycle.
* The various MCUs in the AVR family have different built-in support for communicating with external devices. In this project we will make use of modules for UART, SPI, I2C communication.
* We are also going to use the external memory bus interface to connect an SRAM and other devices.
  + Both are described on Atmel’s product page and datasheets
* We will use minimum two different AVR microcontrollers in this project
  + ATmega162 in node 1
  + ATmega2560 in node 2
  + Optionally the AT90USB1287 on the USB Multifunction card
  1. USB Multifunction card
* The USB multifunction card contain an Atmel AT90USB1287 microcontroller. It is not mandatory to program the MCU, but the card has a lot of feature, so this is great piece of hardware to use in the creation of additional features.
* We are only going to use the joystick, pre—processed touch signals and the OLED display in the exercises
* Some of the onboard features
  + Thumb joystick with button
  + Touch interface
  + Speaker
  + CAN interface

2.17 Arduino mega 2560

* The Arduino is usually programmed through the USB interface using for instances the Arduino IDE. This gives access to a vast number of libraries
* We will use the JTAG interface and Atmel Studio to program the Arduino.
* We will not have access to the Arduino libraries, the code will more closely mirror that in node 1.
  1. Atmel ICE
* Is a powerful development tool that lets you debug programs that are actually running on the AVR microcontroller. Together with Atmel Studio, you can step through code line-by-line, read and manipulate most I/O and control registers, insert breakpoints and so on.
  1. JTAG on the Arduino
* The interface and language used to communicate with an Atmega during debugging is called JTAG. This is one of many programming and debugging standards, most Atmegas support both JTAG and programming via SPI. Called In System Programming (ISP). It is possible to to disable the JTAG and ISP interface to prevent reprogramming of the microcontroller. This is done to through internal settings, or fuses in the microcontroller.
* To be able to use debugging on the Atmega2560 we therefore have to enable JTAG again. This can be done via the ISP interface which is left active.
* Follow the instruction on how to do it in the term project file

2.110 Hardware debugging

* There will be errors and bugs when assembling the breadboard. And that’s completely normal. Before asking for help, go through these debugging steps given in the project assignment file.

Tips

* The ICs must be handled with care.
* Turn of the power supply when working with components
* Use color-coding
  + GND = black
  + Voltage source = red
  + Data bus = colour 1
  + Interrupt signal = colour 2
  + Analog signal = colour 3
  + Etc
* Remember that not connecting a signal is not the same as 0. Unconnected pins will probably float and take random values.
  + Ensure that all pins are either connected to ground or two voltage source.
  1. Programming in C with AVR
* C – programming skill are essential to this project. If you don’t master C sufficiently, you are recommended to buy a book or go through tutorials on he web.
  1. Modularization and drivers
* The code will eventually get very big and it will get difficult to find any errors.
* The code must be broken into logical and manageable modules with clear interfaces.
* A good way to make the code in this project easy to follow and maintain is by dividing it into driver modules where possible.
  + Driver for SPI
  + One for MCP2515
  + One interface for sending and receiving CAN messages
* For each header, make one header file and one code file.
* Avoid unnecessary dependencies and global variables
  1. Documentation
* As far as possible, your variables, functions and defines should have self-explanatory names.
* You should also comments the code you write in such a way that functions without self-explanatory names or hard-to-read algorithms can be understood by other people
* Doxygen is a good online tool that will generate on-line documentation
  1. Ports
* Most pins of the AVRs can be configured by the application software as digital inputs or outputs by way of dedicated control registers. Groups of pins are organized into ports. Each having 8 pins. Atmega 162 has four ports: PORTA, PORTB, PORTC, PORTD
* Each port on the MCU used in this project is controlled by three 8-bit registers; portx, DDRx and PINx (where x is the port name; PORTA).
* Each bit in the registers correspond to one pin on the port.
* DDR registers are used to configure a port as input or output
  + 1 I output
  + 0 is input
* The PORT registers have two functions, depending on the pin being input or output
  + If the pin is an output
    - The port register is used to control the state of the pins: 1 sets a pin to a logical high value, 0 to a logical low.
  + If the port is input
    - The port registers enables an internal pull-up on the pins set to 1
* PIN registers can only be read, and each bit will correspond to the state of the digital input.
* Note that most pins have a shared functionality. That is, other peripheral unit on the microcontroller might override or be overridden by these.
  1. Bit manipulation
* Configuration and control of a microcontroller and its peripherals are usually carried out by manipulation of bits in dedicated control registers. Hence, register oriented programming and bit manipulation comprises an important part of any microcontroller program.
* The header file ‘avr/io.h’ in AVR Libc associated the names and addresses of all AVR registers as well as the bit names and positions for each bit in the various registers.
* A lot of bit manipulations will occur, defining macros can be very useful.
  1. Polling and interrupts
* Embedded systems are required to react within reasonable time to relevant events occurring in their environment and thus need a sufficient mechanism to detect when something has happened. Two techniques exist polling and interrupt
* Polling /busy waiting
  + Prosessor continuously monitors if the event has happened, typically by actively checking for a bit change. Easy to implement, uses a lot of resources.
* Interrupt
  + Are a technique of automatically diverting the processor form the execution of the current program so that it may deal with some event that has occurred. E.g. when the ADC conversion is complete.
  + When an interrupt has occurred, the processor automatically saves its current state and the interrupt service routine (ISR) is called.
  + The prosessor determines the identity of the interrupt in two way, by polling all the interrupts, or by receiving the identity of the interrupt directly from the interrupt device (interrupt vector).
  + The identity code is used to look up the address of the corresponding interrupt service routine in the processor’s interrupt vector table and then a call to this routine will be executed.
  + How to create routines is described in the AVR Libc manual and how to activate the hardware interrupts of an AVR is described in the AVR datasheets
  1. Struct and Union
* Structs enable grouping of variables together as a single entity in an object oriented fashion, but it cannot include associated methods like in a true object oriented language.
* For example, a struct can groups all variables needed for sending a CAN message
* Typedef statement lets you referrer to the struct as a data type subsequently in the program.
  1. Useful libraries
* ‘avr/io.h’
  + AVR device-specific IO definitions
* ‘avr/interrupt.h’
  + Interrupts
* ‘util/delay.h’
  + Convenience functions for busy-wait delay loops
* ‘stdint.h’
  + Standard integer types
* It is recommended to check the AVR Libc documentation and see if there are function in these libraries you may find useful.
* The delay function require the variable F\_CPU to be set to the current clock frequency.
  1. Software debugging