***
Page 1   PCB_111000_CP2102 (or just the PCB):Press a/k to continue.

Learning with PCB111000 is based on playing with simple programs, getting familiar with the way they are written, understanding them and finally modifying them to make them our own. The programs drive an 8 digit display, generating patterns and numbers and pass data to and from a PC.

Before attempting to begin programming in earnest it is a good idea to read the Introduction and User Guide and of course download the applications suggested there.

Our first project a simple C program is based on JP's Blinky, see JP p30 and chapter 3.   A selection is offered here. A free-bee is given, just to make sure that everything is working OK. Just follow the instructions given in the user guide.

Why not choose another and type it in. Verify the code and upload it to the pcb (remember Ctrl z, the most useful go back feature).

System crashes: Every effort has been made to prevent these, but if pulsing the reset control switch does not restore the p/r/t/D prompt reconnect the PCB to the PC with this switch held in.

Having got it working pressing Tools/Auto Format will make the code much clearer to read. Why not try several others and then take a look at them and notice a number of features:

#include "First_project_header.h"   Click on it and see that it contains additional information required by the compiler, including other .h and .c files with essential definitions and resource subroutines (pn is recommended for opening these).

The main routine. This is where the program starts when power is first applied or after a reset.
Move on the page 2 for more features (press a/k).*

*****Page 2**   More things to take note of in our C programs.

setup_328_HW (Hard Ware):
This contains a block of code and is known as (k\a) a macro. Details are given in the header file. HW requiring setup includes
The data pins,
communication between the micros and with a PC
the reset control switch and

watch dog timer(WDT) which automatically reset the micro.

sei() which enables interrupts which are required by the one wire link (and will be discussed later).

Unsigned int PORT_1:

PORT_1 is the name of a memory location that holds numbers. The smallest element of computer memory is a bit and can hold either a 1 or a 0. Memory is often divided into blocks of 8 bits which can hold numbers between 0 and 255. 16 bits are reserved for PORT_1. See JP chapter 4 for an introduction.

Hexadecimal numbers (i.e. 0x8000)

In these 16 in ordinary counting is equivalent to 0x10 and 256 (16 x 16) is equivalent to 0x100.

Arithmetic and logic symbols.

JP gives full definitions of these topics in chapter 4. However logic quickly gets rather involved and a quick look is all that is recommended at this stage.

The for loop, the while loop and the if statement

These are very important features of C. A good look at JP chapter 5 C Control flow is recommended before moving on from this first project.

Subroutines

One_wire_Tx_2_integers(PORT_1, PORT_1)

This sends data over the one wire link to drive the display.

PRN_16bit_GEN(0) which generates random numbers

Timer_T2_10mS_delay_x_m() which uses a HW timer to generate delays.

Note: Subroutines are called by the main routine or other subroutines. Here we define them as either resource subroutines which are for general use or project subroutines which are written for a particular project. See JP page 87 chapter 6 for more on subroutines.

break and continue commands:

These are both executed when a particular condition is met. Break causes the program to exit a for or while loop. Continue causes it to skip a number of lines of code and jump back to the start of the loop. See JP page 79.

SW_reset;

This is also a macro. It uses the watch dog timer (WDT) to reset the program so that it repeats endlessly.

Before leaving example 1 why not comment each line stating what it does as a learning aide (See JP p39).

Having looked at the examples why not mix and match to compose your own code.*

***Page 3**
Talking to the PC, introducing the USART (Universal .... Receiver Transmitter module).

Many devices have a receiver transmitter module for communicating with a PC.
  See JP page 98 chapter 6 for a few details.
Arduino provides a comprehensive set of functions for using the USART.
Here basic do it yourself (DIY) functions are used.
  This approach provides a good introduction to the data sheet, a vital resource for setting up other things such a timers and interrupts.

Take a look at subroutine Char_to_PC_Local() and see section 20.11.1 of the data sheet.
  UDR0 is the input output (I/O) data register for the USART, it has 16 bits, 8 for write operations and 8 for read.
  Data written to UDR0 will automatically be transmitted.
  Data that has been received can be obtained by reading UDR0.

UCSR0A is the Control and Status Register A.
  UDRE0 (bit 5 of register UCRS0A) is a flag which is set when UDR0 is empty and can accept new data for transmission.
  RXC0 (bit 7) is a flag which is set when data has been received and can be read.

A number of these DIY functions can be found in Resources_Receiver_transmitter\Basic_IO_and_Timer. They are best viewed using programmers notepad (pn).*

***Page 4**
Nine sample programs that transfer data to and from the PC are offered.
A look at each is recommended.

Example 1 simply echoes keypresses to the screen.

Example 2 prints many of the common characters to the screen.
  Clicking on the hex button in the receive section of the Bray window shows that each character is represented by a number. See JP p283.

Example 3 prints a file.

Resource "isCharavailable(65)" returns a 1 immediately if a character is received in a 1 second window. Otherwise it returns zero and is called again.
Having received the first character it can then receive any number but will exit (break) if no character is received after a pause of 154mS.

Example 4 introduces character strings and shows they can be sent to the PC.

It also shows how they are stored in arrays (consecutive locations) in memory.
We compose the strings and choose a name for each one.
The compiler adds a 0 to the end of each string.
It also replaces the names (i.e. message_1) with the address of the first character in the string.

Note the two ways of writing the appropriate the C statements: The * character indicates that memory location message_1 holds an address. The [ ] characters indicate that message_3 is an array of characters. See page 5 for more UART examples.*

***Page 5** More examples of data going to and from the PC.

Example 5 simply prints out the characters together with their numeric equivalents.

Example 6 Converts a number to a string of characters and sends them to the PC and example 7 does the reverse. These are two very important operations. Take time to understand what is going on. See JP pages 81 to 84 for some help.

Example 8 Uses project subroutines to: Accept a numerical string from the keyboard, convert this to a number which it doubles, reconvert the result to a string and send this to the PC.

All the strings are stored in array Num_string which has 12 memory locations. The subroutines require the address of the first element in this array (hence the use of the * symbols).

Variables preceded by the * symbol are k/a pointers. See JP ch. 8 for a discussion of pointers.

Example 9 uses repeated division to convert a fraction to decimal.

It calls subroutine divide() which returns the number of outstanding decimal places.

It also calculates the result of the division and the remainder (modulo see JP page 51).

Memory for these two variables is provided by the main routine.

The & symbols indicate that the subroutine will not be providing memory for these variables. It is the address of each variable that is being passed to the subroutine not its value.

As an alternative to the use of pointers (i.e. the * and & symbols), variables can be declared in the header file rather than in the main. This may be OK for small projects but is not normally considered good practice (see JP page 90).*

***__Page 6__ Using interrupts:

5 examples are given. See JP chapter 7 p109 for a general introduction to interrupts.

3A does some calculations and prints out the results when sw2 is pressed.
A HW timer running in the background interrupts this process every 100mS.
The micro jumps to the ISR (interrupt service routine) and updates the display.
A key press interrupt is also shown enabling users to to manipulate the display manually.
We see that the interrupt enables 2 processes to run at once.

Look at the 100mS_clock_tick() subroutine.The timer is configured as a counter which counts from 0 to 12500. TCCR1B sets a timer clock to 125KHz so it takes 100ms to complete counting, raise an interrupt and start again. Open up the data sheet section 16.11 to see the registers. We leave most of them alone but set TCCR1B, TCNT1, OCR1A and TIMSK1.

3B Target practice looks at interrupts generated by pressing one of the user switches.
PCI stands for pin change interrupt.
Look at the header file for definitions of the macros.
See the data sheet sections 13.2.4 and 1 for details of the PCI registers.

3C Reaction time tester:
Nothing new here, just included for a bit if extra practice.
Press SW2 and release immediately that the leds line up to get the display to flash.

3D Multi display:
This includes PCI and timer interrupts as well as the one wire comms interrupt.
Multiple interrupts can easily cause trouble.
To avoid this the PCI also resets the micro.
Before doing so it writes to the EEPROM.
After the reset it reads the EEPROM so that it knows which switch was pressed.
EEPROM is a special area of memory that survives power down and reset, which both usually result in all registers being reset to zero.

3E Is an interrupt driven key press logger that also drives the display.
Projects 3D and E include the volatile declaration.

This is because variables are being changed by an ISR.
Unlike for a subroutine the compiler cannot anticipate that the ISR will be called.
The variables are therefore declared to be volatile as a warning to the compiler.*

***__Page 7__   Exploring logic and binary operations.

Two programs Bit_ops_1 and 2 are used to demonstrate how individual bits can be read, set, reset or changed.
These operations are needed to configure HW items such as the USART, and Timers and also to monitor their status (i.e. is there USART data waiting to be received).

Bit_ops_1 uses the display and random number generator to investigate the AND, OR, Exclusive OR and NOT operations.
Bit_ops_2 combines these operations with the shift function to set or read individual bits.

As a practice exercise in logic, multiple addition and subtraction are used in program_3 to implement multiplication and division (these operations are usually carried out in HW or by compiler generated code).

As a final example the display is used to display positive and negative 8 bit numbers in binary and decimal form.*

***__Page 8__   Using the Arduino PC comms library:

Arduino provide a suite of programs that pass data to and from a PC.

These will probably be the sensible choice for use with the Arduino compiler.

Integer and floating point numbers are supported.

Easy extension to include scientific numbers is implemented in these examples.

Three additional examples are included:
The first explores prime numbers.
The theory for these was developed by the ancient Greeks. Their method for obtaining them is still probably the best for our use here.

The second provides an estimate for pie.
The area of a circle is computed by placing it over a fine grid of squares and counting them.

The third uses a power series to compute log/antilog functions so that any positive scientific number can be raised to an arbitrary power.

If the result is either too large or small the program times out. An interrupt is generated and the resulting ISR, writes to the EEPROM, before issuing a SW_reset. When the program resets it reads the EEPROM and issues the appropriate user prompt.*

***Page 9** The 8 digit display and IO:

Obviously Arduino does not provide library functions to drive the display, so it is necessary to revert to DIY programming.

The purpose of the first 2 programs is to demonstrate how numbers are constructed from the display segments.

Program 3 defines each digit in terms of a string as has previously been done with messages.
The user enters a digit at the PC.
The appropriate string is selected.
Segments defined by the string are are illuminated.

In the remaining projects the complexity of driving the display is handled by the mini-OS that resides on the Attiny1606. Two of these are used to enter data from the PC keyboard and two from the user switches.*

***Page 10** Timers

Here a selection of timers/clocks are developed.

Accuracy is provided by a watch crystal that beats 32768 times a second.
Register TCCR2B (see section 18.1.2) is used to reduce the crystal frequency to 1024 beets per second which are counted by timer/counter T2.
The counter advances by 102 in 99.61mS (~100mS) after which an interrupt is raised.
It advances by 918 in 896.5mS (~900mS).
To ensure accuracy over a 1 second interval the counter must advance by a further 106 (103.5mS) before the 1 second interrupt is raised.

In the first example time is simply recoded as a string

In the second a counter is set up that records elapsed time in seconds.

In the third example much of the time keeping burden is transferred to the mini-OS.

Two more examples are included,
one is a stand-a-lone clock, in which time is entered and adjusted using the user switches and the other is a stop watch.*

***Page 11** Reset options

The HW includes a timer k/a the watch dog timer (WDT).

This counts from zero to a user defined value after which it issues a reset pulse.
Provided that it is reset frequently enough there need be no reset.

Arduino offers three modes for the WDT.
It can be initiated as in the macro "setup_watchdog" (see the project header file). If it is allowed to time out the project will be reset.

It can be initiated with an interrupt (see for example "One_25ms_WDT_with_interrupt").

The ISR (interrupt service routine) might write to EEPROM before issuing a SW_reset.

If the ISR is missing program control will jump to location 0x0000, the start of the program.
This is important for us because a reset takes program control to the boot loader not the start of the user project (0x0000).

PCB111000_CP2102 uses the macro "SW_reset" to implement a planned reset.
It uses "One_25ms_WDT_with_interrupt" with no ISR to recover from an unplanned reset or system crash.

The micro includes an MCU status register (MCUSR).
This records the source of the reset which can be
POR (power on reset)
WDT
External reset (not used here)
Low supply voltage (not used here).

The bootloader reads this to determine whether the reset is due to the WDT or POR
It uses the EEPROM to determine whether to jump to the user program or continue with the bootloader.

Example 8A explores the basic operation of the reset function provided by PCB111000_PC2102

Example 8B sums cosine waves to generate a pulse function.
This is an example of the micro working flat out with an 8MHz clock.
No wonder many applications require a much faster clock.
The various reset functions are used to control its operation.*

***Page 12** That's it for PCB11000_CP2102:

Much ground still to cover, many new things to explore, but hopefully enough ground has been covered to wet the appetite for more detailed and systematic study.

Click on Arduino Help/Reference to see that we have barely scratched the surface of what Arduino has to offer.

There is of course also the Atmel Studio free development system. I would certainly not want to be without either.

Of course the great strength of the UNO is that it comes with a bootloader.   However why not experiment with UNO_AVR_Programmer also posted on AVR freaks.   Even download free PCB design SW and make your own board!*

***Page 13   Not used.*

***Page 14   Not used.*

***Page 15   Not used.*