Jonathan W. Valvano
October 2, 2002, 12-12:50pm

(35) Question 1. Observe PA1, PA0 on a two-channel scope. 00 means `Initialize`, 01 means `Thingy`, 10 means `Stuff` and 11 means `Bailiwick`. The trick is to remember the interrupt suspends foreground execution.

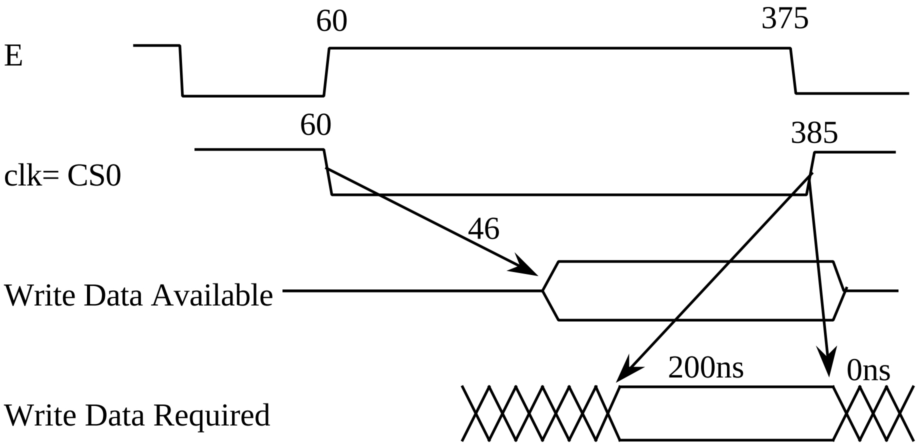
```
void main(void){
  DDRA = 0xFF;      // debugging monitor, scope attached to PA1, PA0
  PORTA = 0;        // 00 means running Initialize
  Initialize();
    while(1){
    PORTA = 0x01;   // 01 means running Thingy
    Thingy();
     if(Mode){
       PORTA = 0x02;   // 10 means running Stuff
       Stuff();
    }
  }
}
#pragma interrupt_handler OC5handler
void OC5handler(void){ unsigned char lastPort;
  lastPort = PORTA;  // save previous setting
  PORTA = 0x03;      // 11 means running Bailiwick
  Bailiwick();
  PORTA=lastPort;    // restore previous setting
}
```

(35) Question 2. Interface an Output chip directly to the MC68HC812A4 data bus
Part a) WDA=(106,t_{cyc}+20)

Part b) WDR = (↑**clk**-200,↑**clk**)

Part c) 106 ≤ ↑**clk**-200, 106 ≤ t_{cyc}+10-200 so 296 ≤ t_{cyc}, so 2 stretches are required.

Part d) Draw the write-cycle timing diagram for the new interface. You may add any additional signals to clarify the system operation.



(30) Question 3. Specify whether each statement is TRUE or FALSE.

True/False

a) A **signed** `char` variable can store values from –128 to +128. E.g.,
`char myData;`
*A **signed** `char` variable can store values from –128 to +127.*

False

b) Consider an **input** device. The **interface latency** is the time from when the software asks for new data until the time new data is ready.
*The **interface latency** is the time from when new data is ready until the time when the software reads the data.*

False

c) Consider an **output** device. The **interface latency** is the time from when the software sends new data to the output device until the time the output operation is complete.
*The **interface latency** is the time from when output device is idle until the time the software writes new data.*

False

d) The `static` qualifier is used with functions to specify the function is permanent, created at compile time and is never destroyed. E.g.,
`static int AddTwo(int in){ return in+2;}`
`static` *this function can only be accessed from this file.*

False

e) The `static` qualifier is used with a variable defined inside a function to specify the variable is permanent, created at compile time and is never destroyed. E.g.,
`void function(int in){ static int myData;`

True

f) The `const` qualifier is used with a global variable to specify the variable should be allocated in ROM. E.g.,
`const int myData=5;`

True

g) Code that is **friendly** means it can be executed by more than one thread with causing a crash or loss of data.
***friendly** means the code only changes what it needs to, and the order of initialization does not matter.*

False

h) Using the `SCI_OutString` and `SCI_OutUDec` for debugging is always **intrusive**.
If SCI12A (interrupt version) is used and the output rate is small enough so that the TxFifo never fills, then these debugging statements are minimally intrusive.

False

i) When interfacing memory to the 6812 in **expanded narrow mode**, PORTD contains the data bus.
PORTC contains the D7-D0 data bus in expanded narrow mode.

False

j) The `volatile` qualifier is used with variables to tell the compiler that code that accesses this variable should be optimized as much as possible.
No, optimization should be turned off when accessing volatile variables

False

k) A read-modify-write access to a shared global variable always creates a **critical** section.
*An atomic read-modify-write access to a shared global variable creates a **critical** section.*

False

l) The compiler automatically places an `sei` instruction at the beginning of the **interrupt service routine** and a `cli` at the end so that the computer runs with interrupts disabled while servicing the interrupt.
No, the 6812 automatically sets I=1 before executing an ISR and the `cli` *instruction resets the I back to 0.*

False

m) In the **linked allocation** scheme described in the Lab 25 assignment, **internal fragmentation** occurs when entire large block is allocated to store a very small file.

True

n) With a linked allocation scheme free blocks may be scattered in physically non-adjacent positions on the disk. Because the entire list of free blocks can always be linked together into one large file, the **linked allocation** scheme has no **external fragmentation**.

True

o) Because the MC68HC812A4 has 22 address lines in extended mode, the DPAGE addressing scheme can access up to 4 million bytes of extended memory.
Even so, DPAGE uses only 20 lines to access up to 1 million bytes of memory.

False