

## Windows error codes

**WinErr: 001 Windows loaded - System in danger**

**WinErr: 002 No undetected errors**

**WinErr: 003 Dynamic linking error - Your mistake is now in every file**

**WinErr: 00F Unexplained error - Please tell us how this happened**

**WinErr: 010 Reserved for future mistakes by our developers**

**WinErr: 019 User error - Not our fault. Is Not! Is Not!**

**WinErr: 01E Timing error - Please wait. And wait. And wait. And wait.**

**WinErr: 01F Reserved for future mistakes of our developers.**

**WinErr: 020 Error recording error codes - Remaining errors will be lost.**

**WinErr: 815 Insufficient Memory - Only 50,312,583 bytes available**

## Lecture 17 objectives

- Basic computer architecture
- Address decoding
- Timing equations (descriptive language)
- Timing diagrams (design and analysis)
- Design steps for interfacing memory to the computer

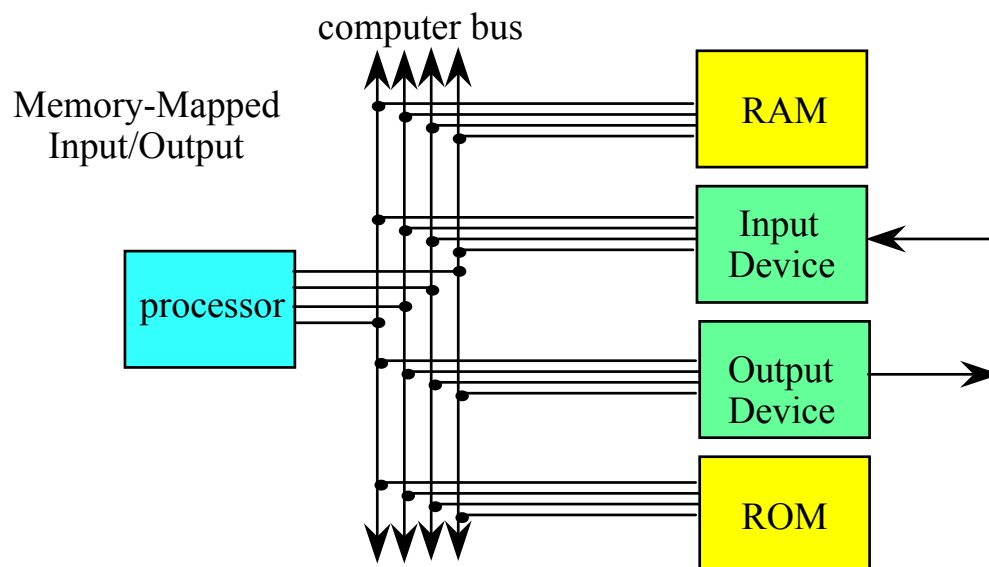


Figure 9.1. Architecture of a computer with memory-mapped I/O.

**Command** part (first of two parts)

A15-A0, specify whether or not the current cycle is meant for our slave  
 R/W, distinguish between the read and write functions  
 generally valid during the second half of the bus cycle,  
 but do not have guaranteed times for its rising and falling edges

**Processor will be the master if no DMA, and determine**

A15-A0, slave address (16-bit)  
 R/W (read=1 or write=0 cycle)

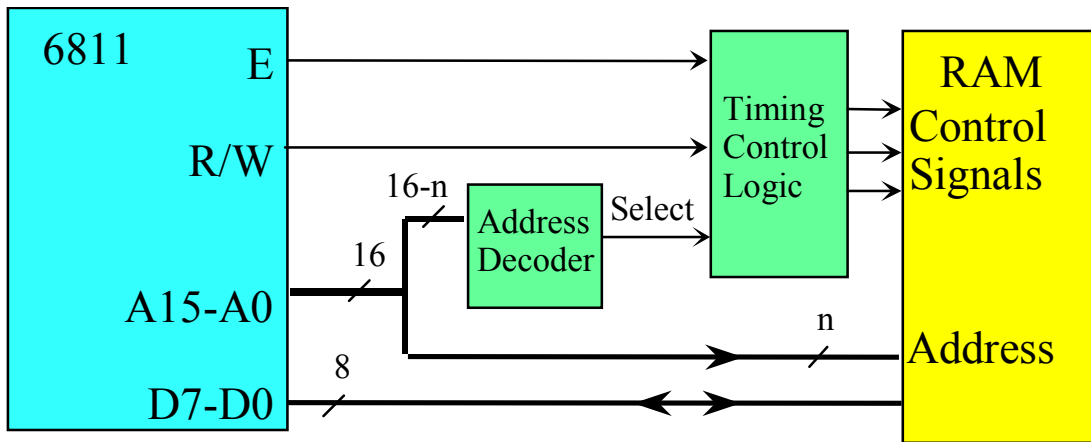


Figure 9.3. Both the 6811 and 6812 support external memory interfaces (expanded mode).

**Select** is true if address matches the slave address.

Select	R/W	Function	Rationale
0	0	Off	Because the address is incorrect
0	1	Off	Because the address is incorrect
1	0	Write	Data flows from 6811 to slave
1	1	Read	Data flows from slave to the 6811

Table 9.1. The address decoder and R/W determine the type of bus activity for each cycle.

**Timing** part (second of two parts)

6811/6812 use a **synchronous** bus  
 rising and falling edges at guaranteed times  
 defines the exact time data is transferred  
 E AS

## 9.2. Address Decoding

determine whether or not the slave has been selected

each slave has its own address decoder

select at most one slave device at a time

**multiplexed** mode, some address pins are the same as the data pins

many microcontrollers have built-in address decoders (6811 does not)

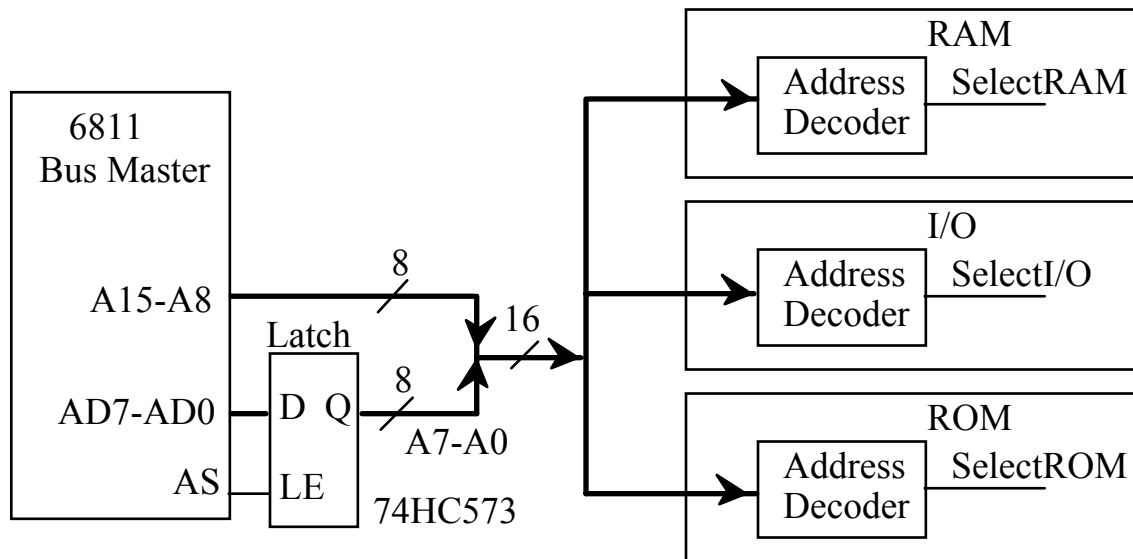


Figure 9.4. The 6811 multiplexes the low address on the same pins as the data.

no two select lines should be active at the same time.

$$\text{SelectRAM} \cdot \text{SelectI/O} = 0$$

$$\text{SelectRAM} \cdot \text{SelectROM} = 0$$

$$\text{SelectI/O} \cdot \text{SelectROM} = 0$$

**Common Error:** If two devices have address decoders with overlapping addresses, then a write cycle will store data at both devices, and during a read cycle the data from the two devices will collide possibly causing damage to one or both devices.

### 9.2.1. Full Address Decoding

Select = 1 if slave address appears on address bus

= 0 if slave address does not appear on address bus

**Example.** Design a fully decoded positive logic

Select signal for an 8K EEPROM at \$E000-\$FFFF.

Step 1.) Write specified address using **0,1,X**, using the following rules:

1. There are 16 symbols one for each of the address bits A15, A14, ..., A0
2. **0** means the address bit must be 0 for this device
3. **1** means the address bit must be 1 for this device
4. **X** means the address bit can be 0 or 1 for this device
5. All the **X**'s (if any) are located on the right hand side of the expression.
6. Let **n** be the number of **X**'s. The size of the memory in bytes is  $2^n$
7. Let **I** be the unsigned binary integer formed from the 16-**n** **0**'s and **1**'s,

$$\mathbf{I} = \frac{\text{beginning address}}{\text{memory size}}$$

In this example:

address= **111X,XXXX,XXXX,XXXX**

beginning address = \$E000

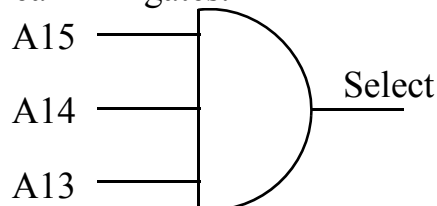
**n** = 13                      size = 8192 bytes = \$2000

$$\mathbf{I} = 111_2 = 7_{10} = \frac{\$E000}{\$2000}$$

Step 2.) Write the equation using all **0**'s and **1**'s. A **0** translates into the complement of the address bit, and a **1** translates directly into the address bit. E.g.,

$$\text{Select} = A15 \cdot A14 \cdot A13$$

Step 3.) Build circuit using real TTL gates.



**Observation:** If we set all the X's to 0 in the row of 0's 1's and X's, we get the starting address.

**Observation:** If we set all the X's to 1 in the row of 0's 1's and X's, we get the ending address.

**Maintenance Tip:** Use full address decoding on systems where future expansion is likely.

**Observation:** We will use symbols to signify negative logic.

**CS1\***  
**/CS1**

### 9.2.2. Minimal Cost Address Decoding

simplify by introducing don't care states for unspecified addresses  
 does not guarantee lowest cost,  
 most embedded systems do not use all the available addresses.  
 unspecified address is one at which there is no device  
 software should never access an unspecified address  
 one and only one device will be selected for valid addresses  
 may select 0,1,or more devices if an unspecified address were to be accessed

Address	Select
Matches our device	true
Matches other specified device	false
Unspecified address	don't care

Table 9.3. Minimal cost decoding optimizes by taking advantage of the don't care states.

**Example. Design a minimal cost select signals in positive logic:**

4K RAM	\$0000 to \$0FFF
Input	\$5000
Output	\$5001
16K ROM	\$C000 to \$FFFF

Step 1.) Write out the addresses in binary for all devices. Include all specified addresses in the computer, not just the devices that we are designing.

RAM	0000,XXXX,XXXX,XXXX
Input	0101,0000,0000,0000
Output	0101,0000,0000,0001
ROM	11XX,XXXX,XXXX,XXXX

Step 2.) Choose as many address lines that are required to differentiate between the devices. Consider the different devices in a pairwise fashion.

Choose A15,A14,A0

The address bits A15 A12 and A0 could also have been used.

Step 3.) Draw a **Karnaugh** map for each device.

- a) Put a true for addresses specified by that device
- b) Put a false for other devices
- c) Put an "X" for unspecified addresses.

**Positive logic** true = 1 and false = 0

**Negative logic** true = 0 and false = 1

Step 4. ) Minimize using **Karnaugh** maps and determine equations

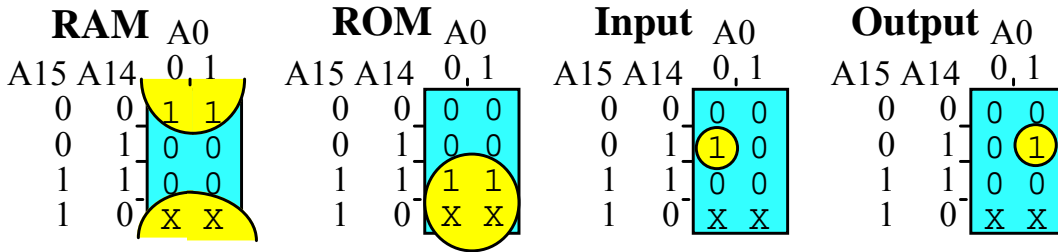


Figure 9.9. Four Karnaugh maps.

$$\text{RAMSelect} = \overline{A14}$$

$$\text{ROMSelect} = A15$$

$$\text{InputSelect} = \overline{A15} \cdot A14 \cdot \overline{A0}$$

$$\text{OutputSelect} = \overline{A15} \cdot A14 \cdot A0$$

Step 5.) Build circuit using real TTL gates:

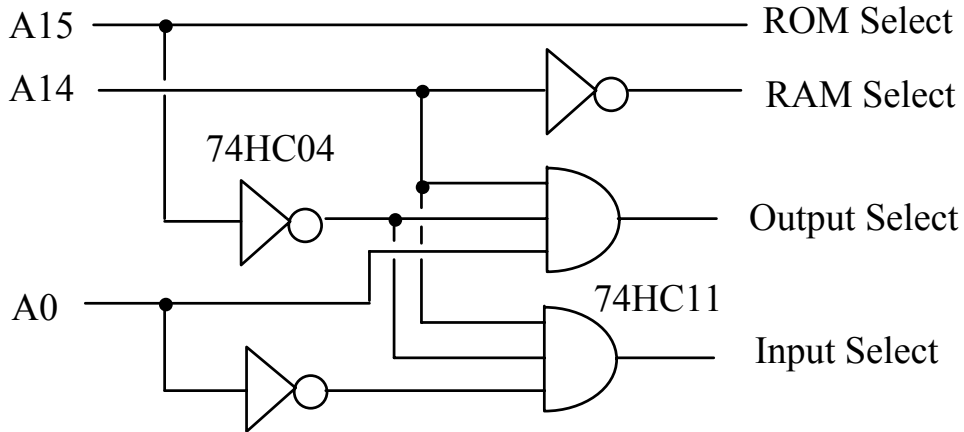


Figure 9.10. Four address decoders like these could also be implemented with programmable logic (PAL).

To implement **negative logic** we can either put true=0, false=1 into the Karnaugh map, or we can build the decoders in positive logic and invert the outputs.

**Performance Tip:** Use minimal cost decoding on systems where cost and speed are more important than future expansion.

**Observation:** If there are no unspecified addresses then minimal cost decoders will be the same as the full address decoders.

**Observation:** If one selects too many address lines, then the Karnaugh map will be harder to draw, but the resulting solution should be the same.

protection can be achieved by making none of the slave circles overlap  
 OK if the software inadvertently accesses the unspecified address  
 “Safer” solution activates only the ROM, if unspecified address is accessed.

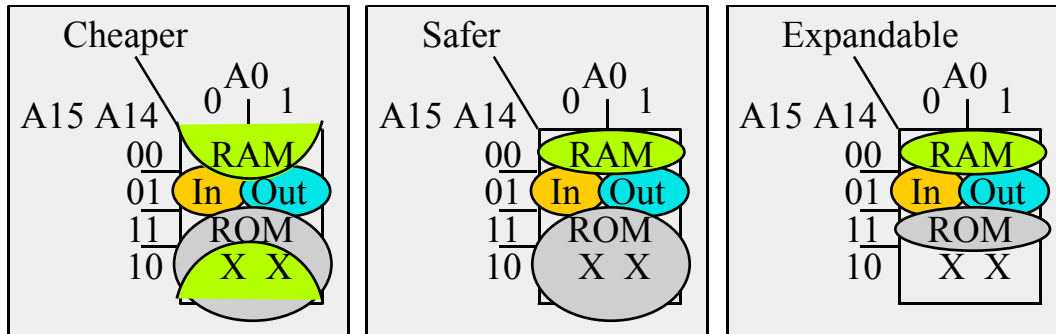


Figure 9.11. As shorthand method for drawing multiple Karnaugh maps.

### Memory map for your 6811 embedded system

\$0000 to \$01FF 512 bytes of internal RAM (globals and stack)

\$1000 to \$103F input/output ports

\$B600 to \$B7FF 512 bytes of EEPROM (your program)

\$BF00 to \$BFFF internal boot loader (to download programs)

**\$E000 to \$FFFF your external 8K EEPROM**

### \*\*\*Lab 8 to do\*\*\* design an address decoder for your 8K EEPROM

*You will first design the decoder equations in positive logic*

*You need to download and read the data sheet for the 74HC138*

*You will implement the interface in negative logic using the 74HC138*

*You need to download and read the data sheet for the 28C64*

## 9.3. Timing Syntax

### 9.3.1. Available and Required Time Intervals.

( 400 , 520 )

( [400,430] , [520,530] )

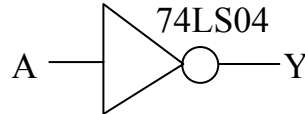


Figure 9.15. A NOT gate.

the time interval when **Y** is high can be written:

$$(\uparrow Y, \downarrow Y) = (\downarrow A+10, \uparrow A+10)$$

If we assume the propagation delay through the 74LS04 can range from 0 to 20ns, then the time interval when **Y** is high can be written:

$$\begin{aligned} (\uparrow Y, \downarrow Y) &= ([\downarrow A, \downarrow A+20], [\uparrow A, \uparrow A+20]) \\ &= (\downarrow A+[0,20], \uparrow A+[0,20]) \end{aligned}$$

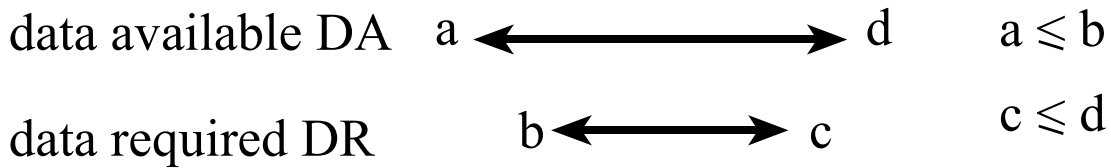


Figure 9.16. The timing of data available should overlap data required.

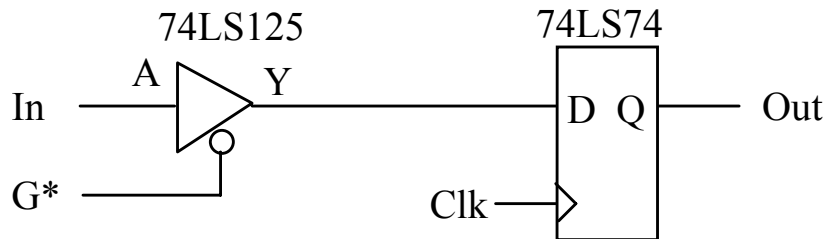


Figure 9.17. Simple circuit to illustrate that the data available interval should overlap the data required interval.

$$DA = (\downarrow G^*+[10,20], \uparrow G^*+[0,15])$$

$$DR = (\uparrow Clk-30, \uparrow Clk+0)$$

Without loss of information, we can write:

$$DA = (\downarrow G^*+20, \uparrow G^*)$$

Thus the data will be properly transferred if

$$\downarrow G^*+20 \leq \uparrow Clk-30 \text{ and } \uparrow Clk \leq \uparrow G^*$$

### 9.3.2. Timing Diagrams

Symbol	Input	Output
	The input must be valid	The output will be valid
	If the input were to fall	Then the output will fall
	If the input were to rise	Then the output will rise
	Don't care, it will work regardless	Don't know, the output value is indeterminate
	Nonsense	High impedance, tristate, HiZ, Not driven, floating

Figure 9.18. Nomenclature for drawing timing diagrams.

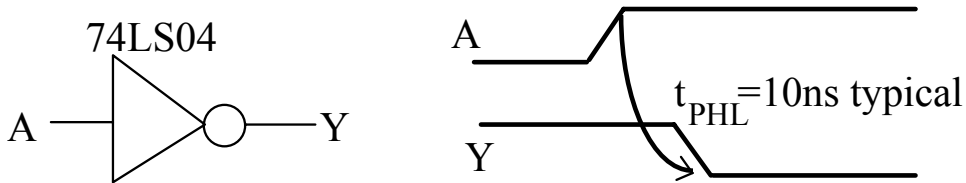


Figure 9.19. Timing of a NOT gate.

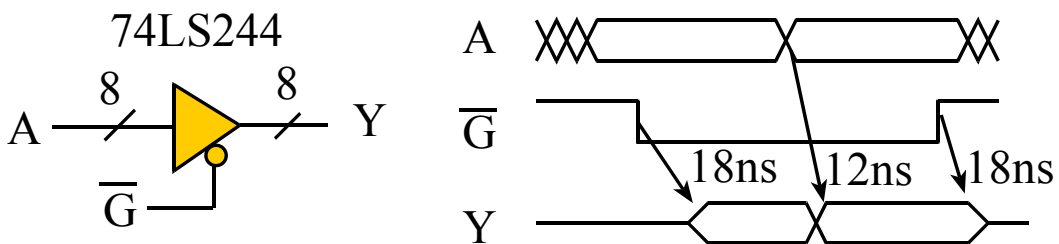


Figure 9.20. Timing of a tristate buffer gate.

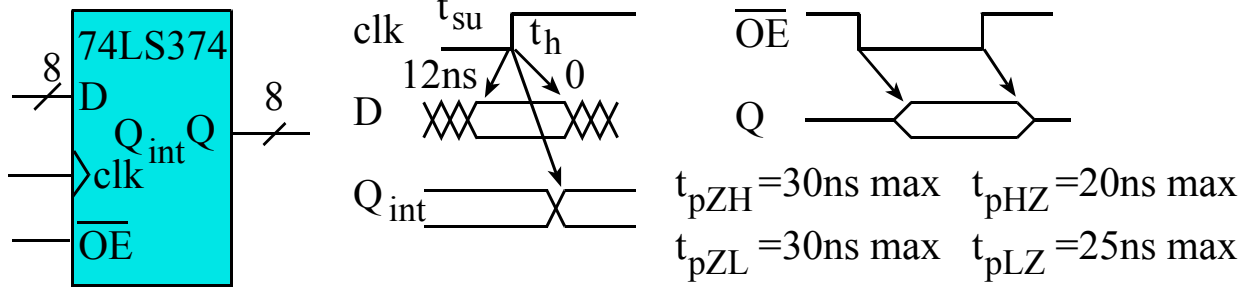


Figure 9.21. Timing of a D flip flop with tristate outputs.