

9.7.6. Extended Address Program Page Interface

Why pages?

- A way to organize large chunks of memory
- Solve external fragmentation
- Implement virtual memory

What is paging on the 9S12C32

- Program Page (PPAGE)
- Up to 1Meg bytes can be accessed
- Memory structure 64 pages of 16k each
- Logical address consists of
 - page number (6 bits) and
 - offset (14 bits)
- Physical address (linear) 0 to \$FFFFFF
 - XAB19:14 (Port K)
 - A15:A0 (Ports A,B)

3.1.9 Program Page Index Register (PPAGE)



Table 4-7 64K Byte Physical FLASH/ROM Allocated

| Address Space | Page Window Access | ROMHM | \overline{ECS} | XAB19:14 |
|---------------|--------------------|-------|------------------|----------|
| \$0000-\$3FFF | N/A | 0 | 0 | \$3D |
| | N/A | 1 | 1 | |
| \$4000-\$7FFF | N/A | 0 | 0 | \$3E |
| | N/A | 1 | 1 | |
| \$8000-\$BFFF | External | N/A | 1 | PIX5:0 |
| | Internal | N/A | 0 | |
| \$C000-\$FFFF | N/A | N/A | 0 | \$3F |

Interface a 1Meg by 8 RAM to the 9S12C32

Timing

- RDA overlaps RDR (no stretches)
- WDA overlaps WDR (needs E --- CS2)

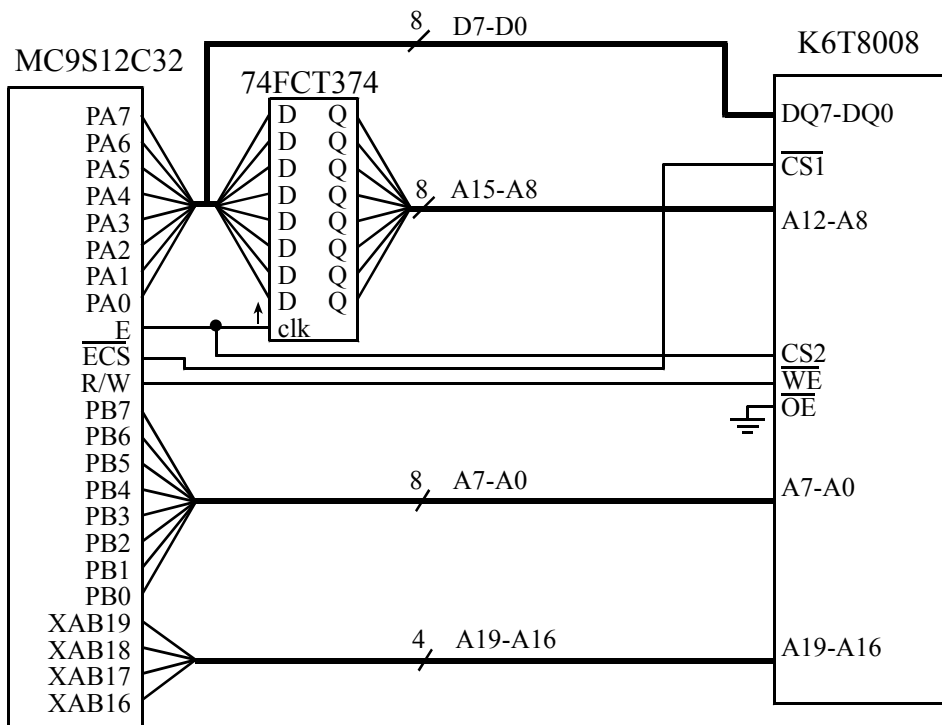
Status part of the design

ECS (PK6) --- CS1

R/W --- WE (OE grounded)

A19-A0 address

D7-D0 data



Read Cycle Timing

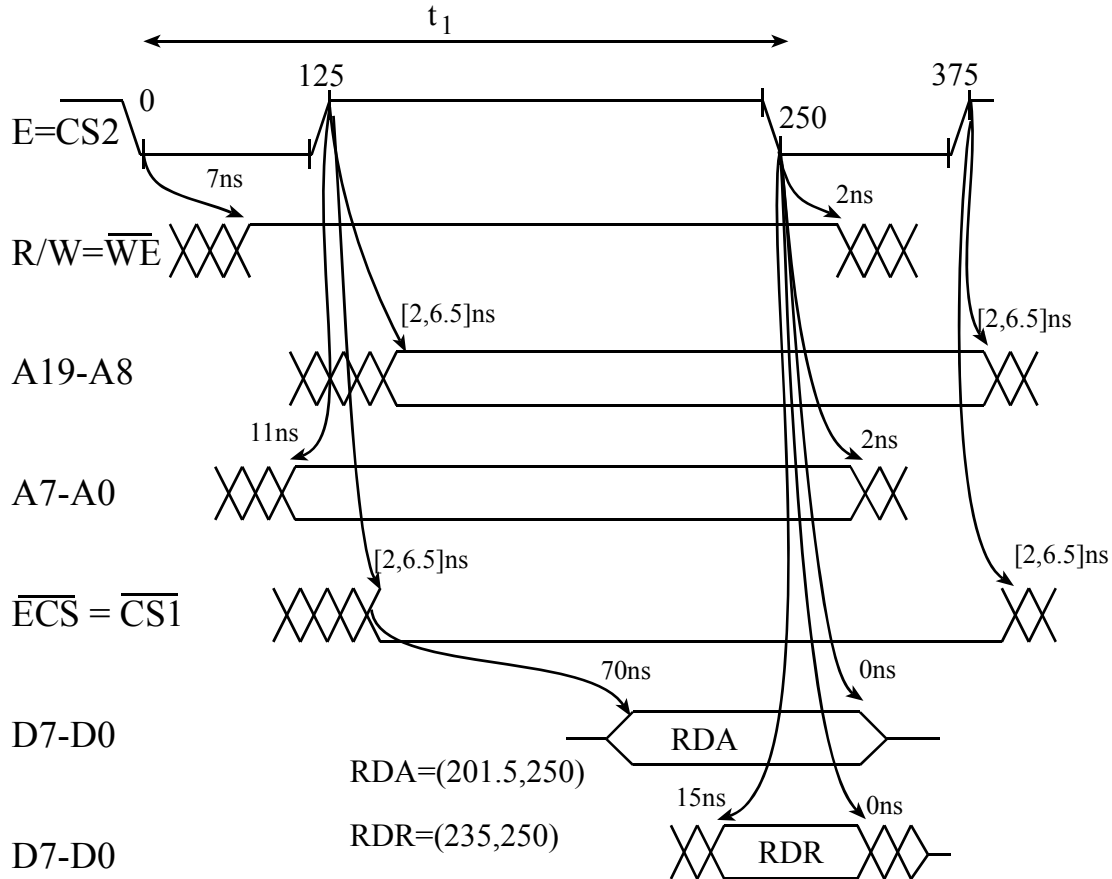
$$\begin{aligned} \text{Read Data Available} &= (\text{later} (\text{AdV}+70 , \downarrow \overline{\text{CS1}}+70 , \uparrow \text{CS2}+70 , \downarrow \overline{\text{OE}}+35) , \\ &\quad \text{earlier} (\text{AdN} , \uparrow \overline{\text{CS1}} , \downarrow \text{CS2} , \uparrow \overline{\text{OE}})) \\ &= (\text{later} (131.5+70 , 131.5+70 , 125+70) , \text{earlier} (\downarrow \mathbf{E}+2 , \downarrow \mathbf{E}+126.5 , \downarrow \mathbf{E})) \end{aligned}$$

Recall that

$$\text{RDR} = \text{Read Data Required} = (\downarrow \mathbf{E} - 15 , \downarrow \mathbf{E})$$

Thus,

| | | | | |
|------------------|-----------------------------------|-----|----------------------|---------------------|
| Address | $131.5+70 \leq \downarrow E - 15$ | and | $\downarrow E+2$ | $\geq \downarrow E$ |
| $\overline{CS1}$ | $131.5+70 \leq \downarrow E - 15$ | and | $\downarrow E+126.5$ | $\geq \downarrow E$ |
| CS2 | $125+70 \leq \downarrow E - 15$ | and | $\downarrow E$ | $\geq \downarrow E$ |



Write Cycle Timing

$$WDA = \text{Write Data Available} = (\frac{1}{2}t_{cyc} + 7, \downarrow E+2) = (132, \downarrow E+2)$$

Since we must synchronize CS2 to the E clock,

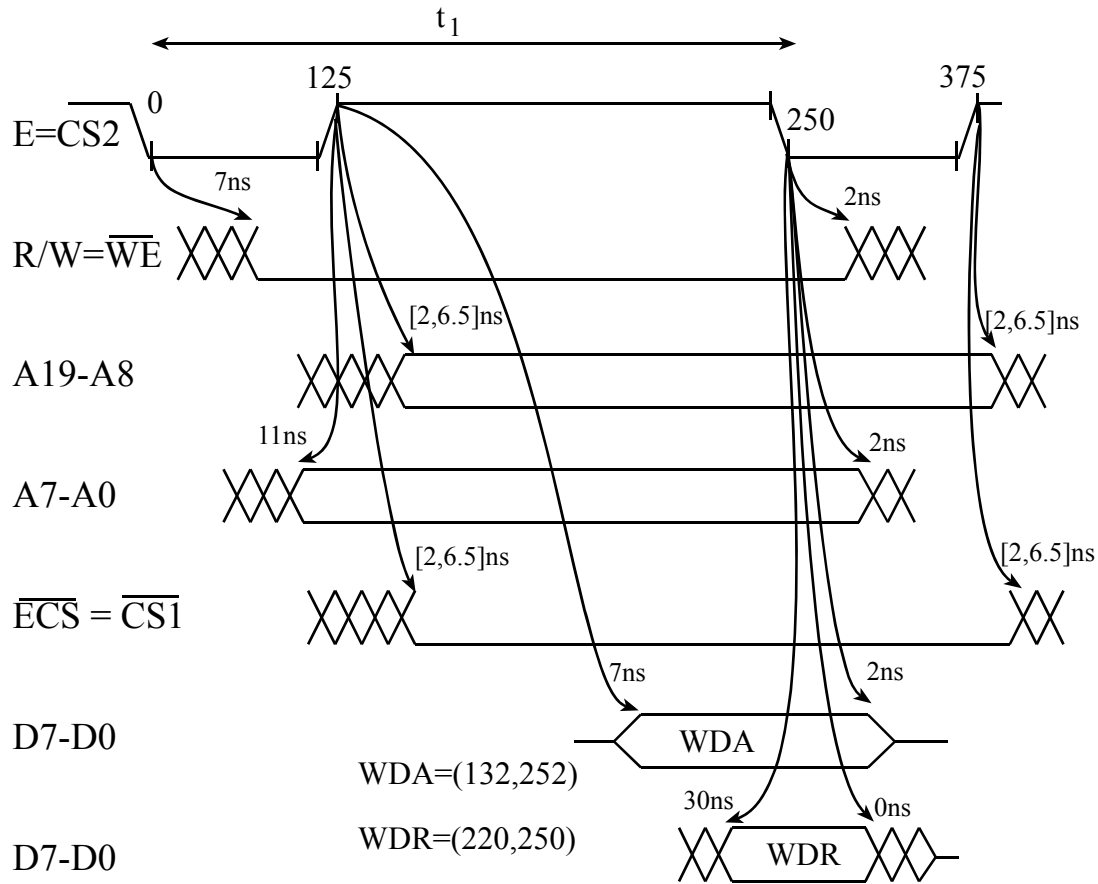
$$\text{Write Data Required} = (\downarrow E - 30, \downarrow E)$$

The worst case delay selects the largest WDR interval

$$132 \leq \downarrow E - 30 \quad \text{and} \quad \downarrow E \leq \downarrow E+2$$

Thus,

$$162 \leq \downarrow E$$



Let A19-A0 be the desired 1Meg memory location.
 To access that location requires two steps
 set most significant addr A19-A14 into PPAGE
 access \$8000 to \$BFFF, least significant addr A13-A0

```

struct addr20
{ unsigned char msb;    // bits 19-14
  unsigned short lsw;  // bits 13-0
};
typedef struct addr20 addr20Type;
char Mem_Read(addr20Type addr){ char *pt;
  PPAGE = addr.msb;    // set address bits 19-14
  pt = (char *) (0x8000+addr.lsw); // set addr bits 13-0
  return *pt;         // read access
}
    
```

Program 9.11. Method for accessing extended memory.

```
void WriteMem(addr20Type addr, char data){ char *pt;
  PPAGE = addr.msb; // set address bits 19-14
  pt = (char *) (0x8000+addr.lsw); // set addr 13-0
  *pt = data; // write access
}
```

Program 9.9. Method for accessing extended memory.

When the software wishes to call a function that exists within the program paged system it uses the call op code. This instruction pushes the old PPAGE and PC on the stack, and loads a new PPAGE and PC as needed. The rtc instruction pulls the old PC and PPAGE from the stack. The effective address of the call instruction includes the msb (8 bits), lsw (14 bits) components similar to the above C programs. One of the most flexible ways to implement the program division is to use a jump table scheme and write relocatable code. Notice the software can use call to execute a function within the same page. Each of the following columns can be assembled separately, using the common equ definitions. E.g.,

```
fun1 equ 0
fun2 equ 3
fun3 equ 6
```

| | | |
|--|--|---|
| <pre>*****regular***** org \$C000 main lds #\$C00 * func2 in page1 call fun2,1 * func3 in page2 call fun3,2 * func1 in page1 call fun1,1 global rts org \$FFFE dc.w main</pre> | <pre>*****page 1***** lbra func1 lbra func2 lbra func3 func1 * func2 in page 2 call fun3,2 rtc func2 * global function jsr global rtc func3 * func2 same page call fun2,1 rtc</pre> | <pre>*****page 2***** lbra func1 lbra func2 lbra func3 func1 rtc func2 * a local function bsr local rtc func3 * func2 same page call fun2,2 rtc local rts</pre> |
|--|--|---|

Program 9.12. Method for calling functions in extended PROM.

Observation: It is important to always use call to execute a function that ends with rtc.

Observation: It is important to always use bsr or jsr to execute a function that ends with rts.