7-Segs

Version 1.1

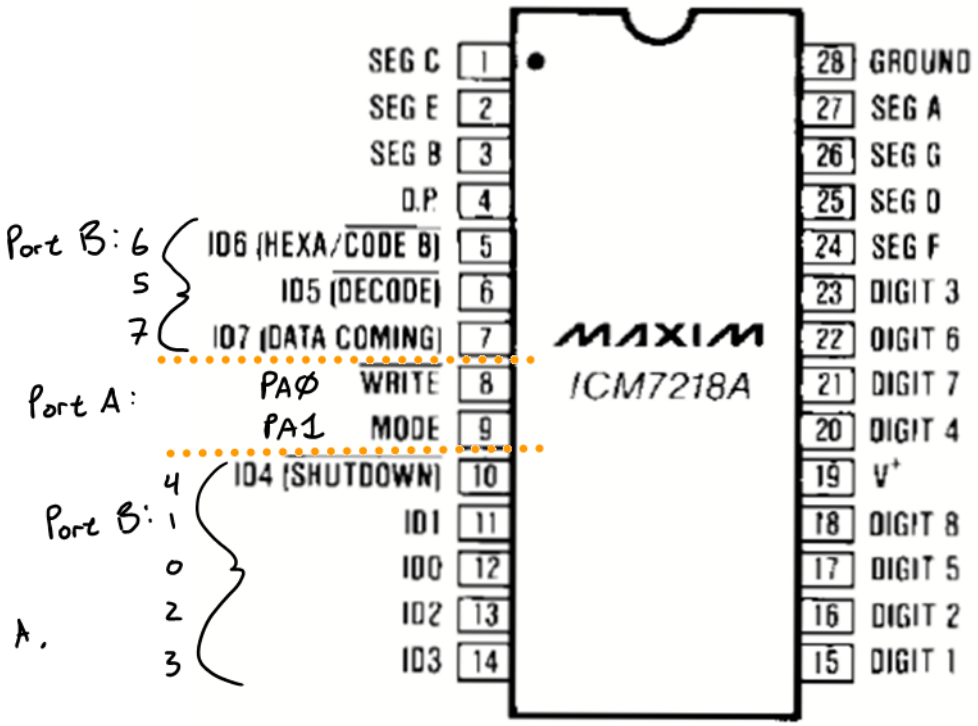
Resources:

- "old" notepack p53→61
- sample 'segs' header
- data sheet for ICM7218

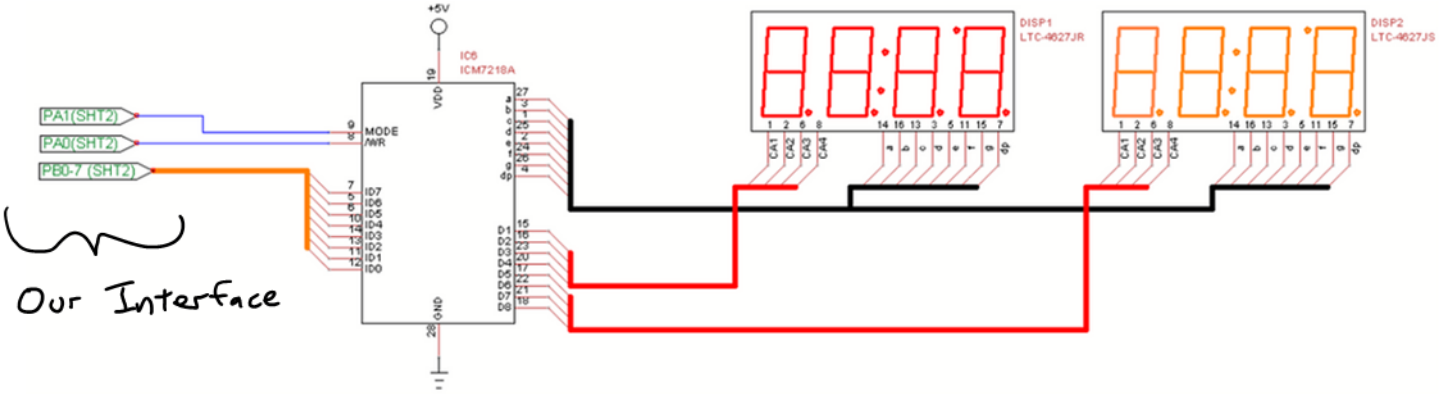
From the ICM7218A Datasheet.

Note weird data order!

This device has been connected to port B & part of port A.



Better: From High-resolution schematics.



We only write to this device. It is fast, and we want that, as we will be using it for diagnostic + regular use. We will rig the library for speed where possible.

Pick 1.5 → Speed / Readability / Convenience

To use this device, we must send a "control" byte and a "data" byte. Normally this device would be connected to a standard data bus, but we will create one synthetically with GPIO. Port A will be used to generate control signals. Port B will be used to send command/data bytes.

Table 1. Input Definitions, ICM7218A and ICM7218B
Note: Pin Configurations for the ICM7218A/B are shown on last page.

	INPUT	PIN	STATE	FUNCTION
PA0	WRITE	8	High Low	Input Not Loaded Into Memory Input Loaded Into Memory
PA1	MODE	9	High Low	Loads Control Word on WR Loads Input Data on WR
PB0	ID0–ID2, DIGIT ADDRESS	12, 11, 13	High Low	Loads "one" Loads "zero"
	ID3, BANK SELECT	14	High Low	Select RAM Bank A (Hex or Code B Select RAM Bank B Data only)
	ID4, SHUTDOWN (MODE High)	10	High Low	Normal Operation Shutdown
	ID5, DECODE/NO DECODE (MODE High)	6	High Low	No Decode Decode
	ID6, HEX/CODE B (MODE High)	5	High Low	Hexadecimal Decoding Code B Decoding
	ID7, DATA COMING (MODE High)	7	High Low	Data Coming (control word) No Data Coming (control word)
PB7	ID0–ID7, INPUT DATA (MODE Low)	5–7, 10–14	High Low	Loads "one" (Note 1) Loads "zero" (Note 1)

Note 1: A "zero" or low level on ID7 turns ON the decimal point. In the NO DECODE mode, a "one" or high input turns ON the corresponding segment, except for the decimal point which is turned OFF by a high level on ID7.

Our Connections:

The "Mode" signal determines if we are sending a command (1) or data (0).
The device "pays attention" when the "Write" signal is low (0).
Since this device is write only, all port pins will be outputs.
During initialization, we don't want the device to be activated.
What would port initialization look like then?

You may not know the port names yet, but what needs to be done (if you recall from the switch/LED library)?

This is Segs_Init()

```
// ensure port pins high
PORTA |= 0x03;

// configure direction of port pins as outputs
DDRA |= 0x03;

// configure direction of port pins as outputs
DDRB = 0xff;
```

(Write Mode) ← need this signal high or device will attempt to do something!

} all pins outputs! (only alter states for pins we are using)

Note: no digital input register for these ports!

We will be raising and lowering MODE and WRITE a lot. Normally we would put such actions in a function, as they are being reused, and this makes code much less prone to error. We want speed here, so we will use macros instead. The macro will cut/paste the code in each place it is encountered, so the code will take more space, but will not need to setup/call a function. This is a speed/space tradeoff.

```
//////////////////////////////////////////////////////
// local helpers
//////////////////////////////////////////////////////
#define Segs_WLATCH PORTA &= (~0x01); PORTA |= 0x01;
#define Segs_ML PORTA &= (~0x02);
#define Segs_MH PORTA |= 0x02;
```

Put this @ the top of the C file. They will only be used by the implementation functions, so they go here.

These are <u>not</u> functions. They are symbols that will be literally replaced with the code at compile time. Do <u>not</u> put function call brackets on them!

Using macros in this way is the best of 2 worlds:

- speed (function call not used)
- single point of failure, or single point of implementation: if it works, it works everywhere.

To do a "normal" write to this device, you send out a command byte then a data byte. As seen, the MODE signal determines if the byte is command or data. You present the command or data byte on port B, then lower and raise WRITE (known as a "write strobe"). When the device "sees" a write strobe, it latches the data on port B, and uses the state of the MODE signal to interpret it.

The command byte tells the device where to put a digit, and how to decode it. The data byte is the data to decode/display.

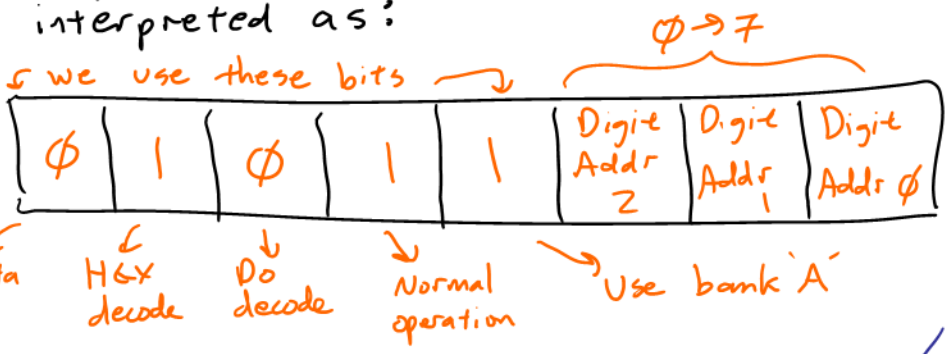
A command byte uses individual bits to determine functions

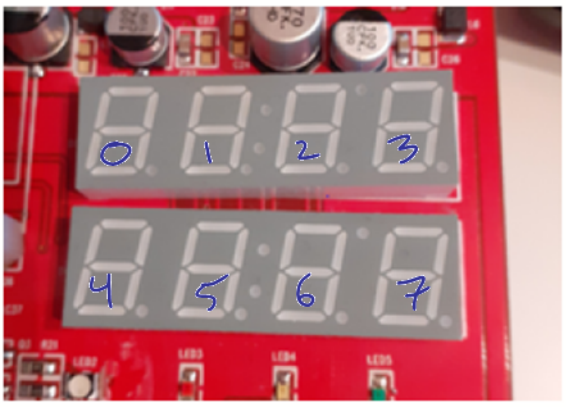
Defaults to use for "normal" output!

Target digit addr

	WHEN MODE IS High		
ID0–ID2, DIGIT ADDRESS		High Low	Loads "one" Loads "zero"
ID3, BANK SELECT		High Low	Select RAM Bank A Select RAM Bank B (Hex or Code B Data only)
ID4, SHUTDOWN (MODE High)		High Low	Normal Operation Shutdown
ID5, DECODE/NO DECODE (MODE High)		High Low	No Decode Decode
ID6, HEX/CODE B (MODE High)		High Low	Hexadecimal Decoding Code B Decoding
ID7, DATA COMING (MODE High)		High Low	Data Coming (control word) No Data Coming (control word)

During a command sequence, MODE must be high, and the data presented will be interpreted as:



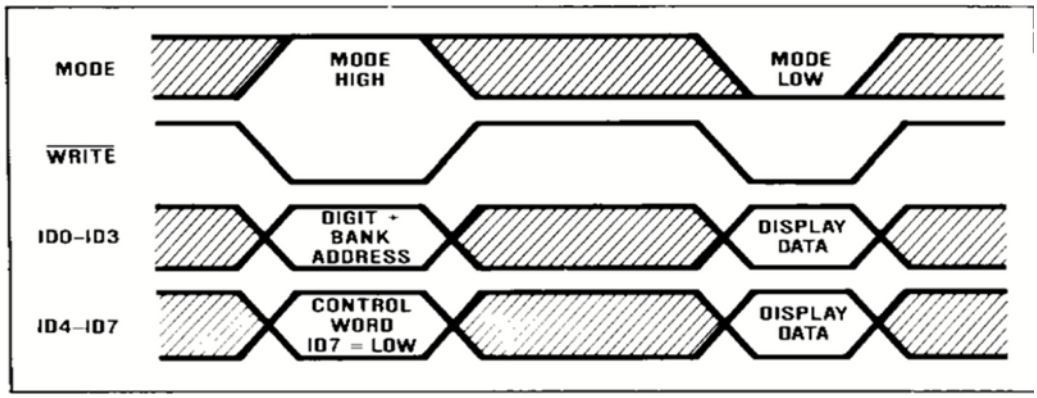


Where is this digit going?

(we only write 1 digit)

Once the command byte is sent (and latched), we write the data byte (and latch it). The data is the value to display as a single digit. In normal decode as HEX, it may be $\emptyset \rightarrow 15_{10}$ ($\emptyset \rightarrow F_{16}$) or $\emptyset x 3\emptyset - \emptyset x 3F$. The device takes literal values or the ASCII codes for values.

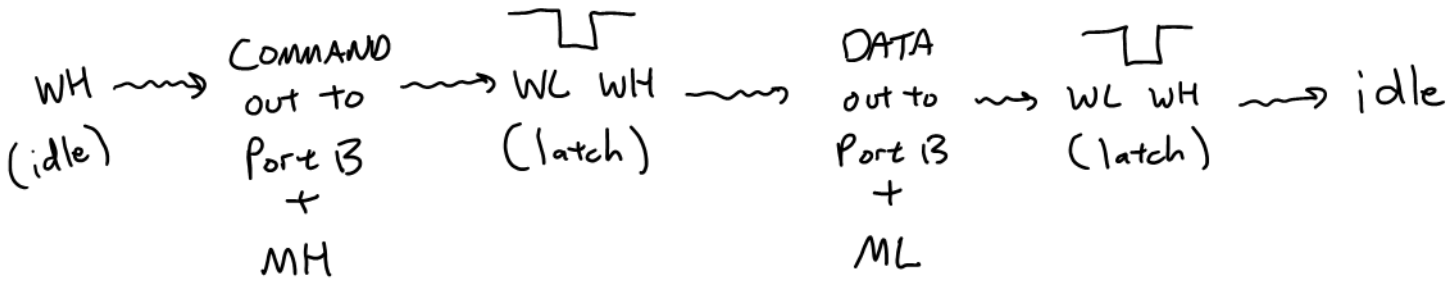
To make the whole sequence work, we must follow the following timing diagram:



You will *always* leave WRITE HIGH when idle.

Figure 5. Single Digit Update Timing—ICM7218A/B

A full digit update cycle looks like this:



Command for next digit

The digit

The Segs_Normal function in the sample header is intended to do all of this, with some argument sanitization:

```
// normal decode, 8-bit value, to address, w or w/o decimal point
void Segs_Normal (unsigned char Addr, unsigned char Value, Segs_DPOption dp)
{
  // trim address to range
  Addr &= 0x07;

  // Bank A, Normal Op, Decode, Hex, No Data Coming
  Addr |= 0b01011000;

  if (dp)
    Value &= (~0x80);
  else
    Value |= 0x80;

  // present control byte
  PORTB = Addr;
  Segs_MH

  Segs_WLATCH

  // present data
  PORTB = Value;
  Segs_ML

  Segs_WLATCH
}
```

Should be 0→7 to place digit on correct element. (keep in range)

See bits for rest of command.

decimal point must be suppressed! (MSB kills DP)

Present command w/ MODE high.

latch command

Present data w/ MODE low.

latch data

Example:

```
Segs_Normal (1, '3', Segs_DP_OFF);
Segs_Normal (3, 12, Segs_DP_ON);
Segs_Normal (6, 0x0F, Segs_DP_OFF);
```

3 in pos 1 (no dp)

12=0x0C C in pos 3 w/dp

F in pos 6 (no DP)

With the "Normal" function, you may write several other functions to place data values of various sizes @ various positions. We will discuss some of these a little later on.

Segs_16D ← decimal (we can't do this yet)

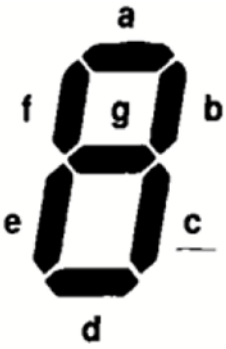
Segs_16H ← HEX, 16-bits, upper or lower "line" of display)

Segs_8 ← Hex, 8-bit (2 digit), any address.

With only a very minor Δ to this function, you may implement the "custom" or "no decode" version (Segs_Custom).

In this function, the command byte does not indicate decode. Without decode, each bit of the data byte represents an individual segment of the digit display:

Data Input	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
Controlled Segment	Decimal Point	A	B	C	E	G	F	D



Segment Assignments

In the command byte, you would take the option For:

- BANK A
- NORMAL OPERATION
- NO DECODE
- HEX DECODE ← LEAVE THIS ON
- NO DATA COMING

The data byte will be interpreted as segment assignments.

```
Segs_Custom (4, 0b10100101);
Segs_Custom (6, 0b00000000);
Segs_Custom (3, 0b11100110);
```

Note: The DP is still backwards (1 = off)!

With this function, you may create a cleared digit, or custom characters (sort of).

Segs_Clear() ← and add to init?