PAT 451/551 INTERACTIVE MEDIA DESIGN I

SENSORS_TO_MAX

NUMBER SYSTEMS

- In any number system, a number represents the sum of a set of digits, each multiplied by successive powers of a base.
- We are accustomed to working with **decimal** numbers, which use the **base 10**
- Any number system of base **b** needs **b** different characters to represent its digits
  - In decimal, we have 0–9

$$5\ 2\ 3\ 7$$

A decimal number

$$* \quad * \quad * \quad *$$

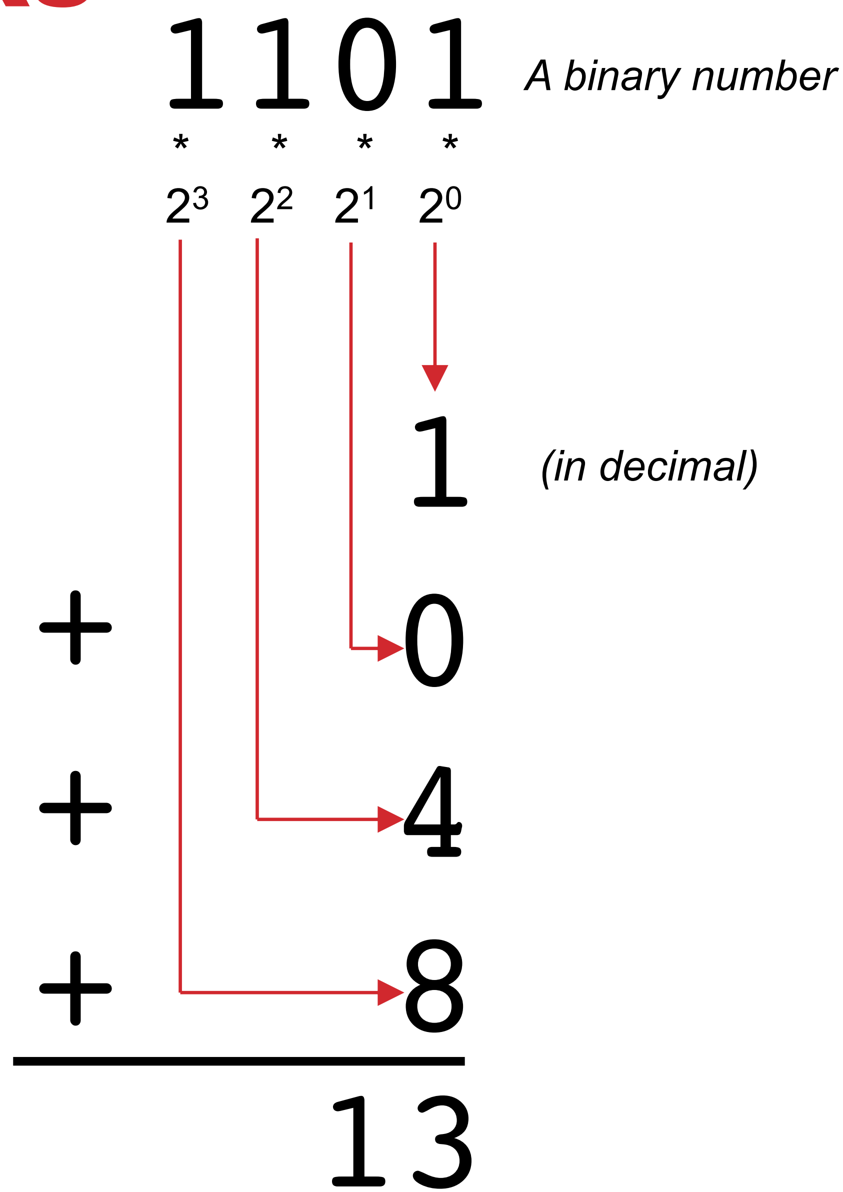
$$10^3 \quad 10^2 \quad 10^1 \quad 10^0$$

Successive powers of 10

$$\begin{array}{r} 7 \\ +\ 30 \\ +\ 200 \\ +\ 5000 \end{array}$$

BINARY NUMBERS

- The term **bit** is a contraction of: **b**inary dig**it**
- Binary is the **base 2** number system
- Digits are 0 and 1
- As in decimal, a binary number is the sum of each digit multiplied by successive powers of 2



NUMBER SYSTEMS

- Recall what we said last lecture about quantization:
- With **n** bits, we can represent 2^n different values, with a range of 0 to (2^n-1)
- Note that the same is true of decimal numbers: with n places, we can represent 10^n values from 0 to (10^n-1)
  - With n=1 places: 0 to 9
  - With n=4 places: 0 to 9999
- In decimal, the maximum value for a given number of places is all 9s
- In binary, the maximum value for a given number of bits is all 1s

1111 1111 *8-bit number*

$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ * & * & * & * & * & * & * & * \\ 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 128 + & 64 + & 32 + & 16 + & 8 + & 4 + & 2 + & 1 \end{array}$$

(in decimal)

$= 255$

SENDING ANALOG DATA OVER THE SERIAL PORT

- We have been using `Serial.print()` and `Serial.println()` to send data over the serial port to the Arduino Serial Monitor.
- These functions format data according to the [ASCII protocol]
- ASCII is a code: 8-bit numbers (0-255) are used to represent different characters, symbols, and typographical instructions.

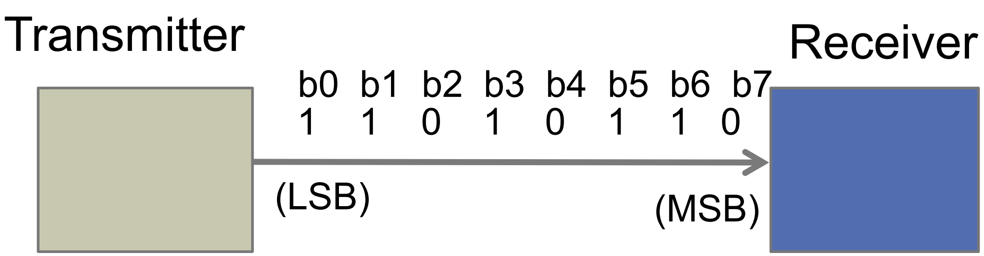
ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

First 127 values of the ASCII table

SENDING ANALOG DATA OVER THE SERIAL PORT

- ASCII was designed with Serial communication in mind: many serial protocols (including the Arduino) only allow sending 8 bits of data (1 byte) at a time.
- **Arduino's Serial protocol can only send 8-bit numbers.**



- When we use `Serial.print`() and `Serial.println`(), whatever data we pass to those functions, whether strings, floats, or integers, ***Arduino sends the ASCII codes that represent that data***

FOR EXAMPLE:

Actual Data Transmitted

	Decimal	Binary	Note
`Serial.println(147);`	49	00110001	*Ascii code for '1'*
	52	00110100	*Ascii code for '4'*
	55	00110111	*Ascii code for '7'*
	13	00001101	*Ascii code* **CR** *(Carriage Return/Enter)*

	Decimal	Binary	Note
`Serial.print("Fred");`	70	01000110	*Ascii code for 'F'*
	114	01110010	*Ascii code for 'r'*
	101	01100101	*Ascii code for 'e'*
	100	01100100	*Ascii code for 'd'*

SENDING ANALOG DATA OVER THE SERIAL PORT

- When we send sensor data from analog inputs to Max or other programs, it is easier to deal with raw data than with ASCII.
- If we were to receive ASCII, we'd have to convert it back to a number, which is not trivial.
- **Arduino's has a method for sending raw data over the serial port:**

  `Serial.write()`

- With `Serial.write()`, the Serial port sends exactly what you pass it:

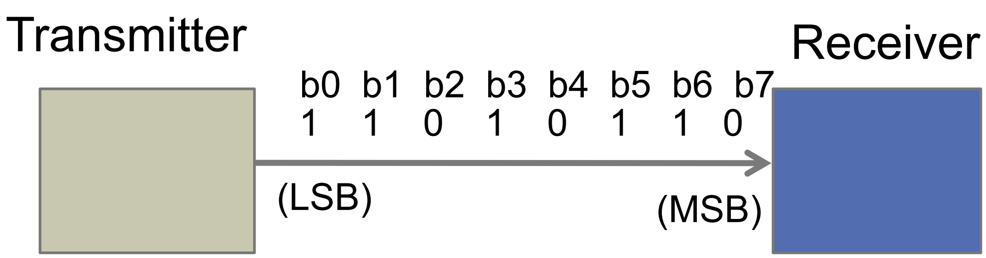
	Decimal	Binary
`Serial.write(147);`	147	10010011

SENDING ANALOG DATA OVER THE SERIAL PORT

- So, for analog data, we can use

```
int reading = analogRead(0); //range of 0-1023
Serial.write(reading);
```

- But remember that Arduino's serial protocol can only send 8-bit numbers in the range of (0-255).



- So what do we do if `reading` is greater than 255?

SOLUTION 1

- We can just linearly scale our analog data to the range of 0-255
- Easiest is to divide by 4.

```
int reading = analogRead(0); //range of 0-1023
Serial.write(reading/4); //range of 0-255
```

- Division is inefficient in microcontrollers, so we prefer to use **bitwise** operations. Right-shifting by 2 places is equivalent to integer division by 4.

```
int reading = analogRead(0); //range of 0-1023
Serial.write(reading>>2); //range of 0-255
```

BIT SHIFTING EXAMPLE

Decimal Number	Binary Number
`876`	`11 0110 1100`

Now, right shift by two:

```
(876>>2)     1101101100
   ==                        (Last 2 bits are truncated)
  219          11011011
```

SOLUTION 1 (14-BIT VERSION)

- If we use 14-bit resolution for analogRead, we need to bit shift by 6 to get an 8 bit number

```
analogReadResolution(14);
int reading = analogRead(0); //range of 16383
Serial.write(reading>>6);    //range of 0-255
```

SOLUTION 2: DESIGN A PROTOCOL

- We can preserve all 10 bits by splitting the original value into two smaller numbers, and reassembling at the receiving end.
- Use bit-shifting and the binary AND (&) to create a **bit-mask**
- Let's split into a 7-bit number and a 3-bit number:

```
int reading = analogRead(0); //range of 0-1023
int b1 = reading >> 3; //7 msb: range of 0-127
int b2 = reading & 7; //3 lsb: range of 0-7
                      // 7 == 0b00000111
Serial.write(b1);
Serial.write(b2);
```

- At the receiving end, we will take the first byte and shift it back 3 places to the left, then add back in the second byte:

```
result = (b1<<3) + b2;
```

TRANSMITTER END

Decimal	Binary
876	11 0110 1100

b1 : Right shift by three:

```
(876>>3)    1101101100
  ==
  109          1101101
```

Last 3 bits are truncated. If input is 10 bits, result is guaranteed to be < 128

b2: & with binary 111 (dec 7):

```
  876       1101101100
&   7     & 0000000111
   ==       ==========
    4       0000000100
```

Only last 3 bits are preserved. Result guaranteed to be < 8

DECOMPOSING A 10-BIT NUMBER

- We know how to get the 7 most significant bits of a number. It's just:

```
(n >> 3)
```

- To get the 3 least significant bits, we use the binary AND operator & and a **BIT MASK**

- The binary AND operator works like this

```
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

- So, if we want to retain only certain bits of a number, we can AND that number with 1s in the position of the bits we want.

DECOMPOSING A 10-BIT NUMBER

- Therefore, if we want the 3 least significant bits, we can & the number with binary 111
- For example:

```
   0110 1011
&
   0000 0111
------------
   0000 0011
```

- In C code, we could write something like:

```
int reading, msb, lsb;
reading = analogRead(0);
msb = reading >> 3; // 7 msb of a 10-bit number
lsb = reading & 7;  // 3 lsb of a 10-bit number
                    // (111 in decimal is 7)
```

REASSEMBLE AT THE RECEIVING END

Decimal	Binary
b1 = 109	b1 = 1101101
b2 = 4	b2 = 100

b1: **Left** shift by three:

```
(109<<3)          1101101
      ==
     872       1101101000
```

Missing 3 bits become 0

b2: Just add the last 3 bits

```
   872       1101101000
+    4   +   0000000100
    ==       ==========
   876       1101101100
```

BUT….

- The receiver may start listening at an arbitrary time

  What if they receive b2 before b1??

- Note that we can't use the value to determine if a given byte is b1 or b2. At any given moment, they could be in the same range.

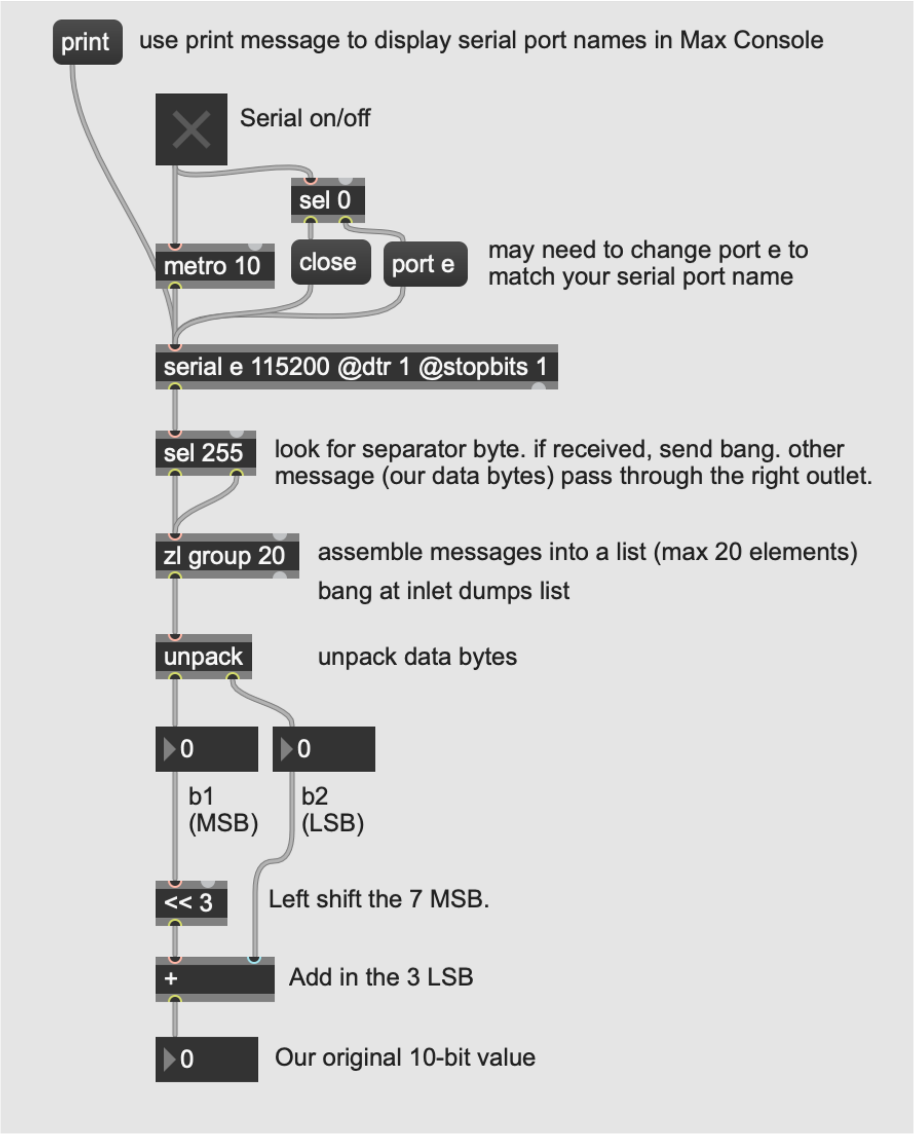
SOLUTION 2B

- Use a “separator” or “marker” or “status” byte that is never going to appear in your byte stream, i.e., any number between 128-255
- Use this “***status byte***” to signify that the next 2 bytes received will be b1 and b2 in that order

```
int reading = analogRead(0); //range of 0-1023
int b1 = reading >> 3; //7 msb: range of 0-127
int b2 = reading & 7; //3 lsb: range of 0-7

Serial.write(255);
Serial.write(b1);
Serial.write(b2);
```

Max Code for Parsing Our Serial Protocol



SOLUTION 2C

- If we want to extend to multiple analog inputs, we can send our "status byte" follow by 2 bytes per sensor, e.g.:

255	(status)
a0_b1	(7 msb for analog input 0)
a0_b2	(3 lsb for analog input 0)
a1_b1	(7 msb for analog input 1)
a1_b2	(3 lsb for analog input 1)
…	
an_b1	(7 msb for analog input n)
an_b2	(3 lsb for analog input n)

SOLUTION 2C - CODE

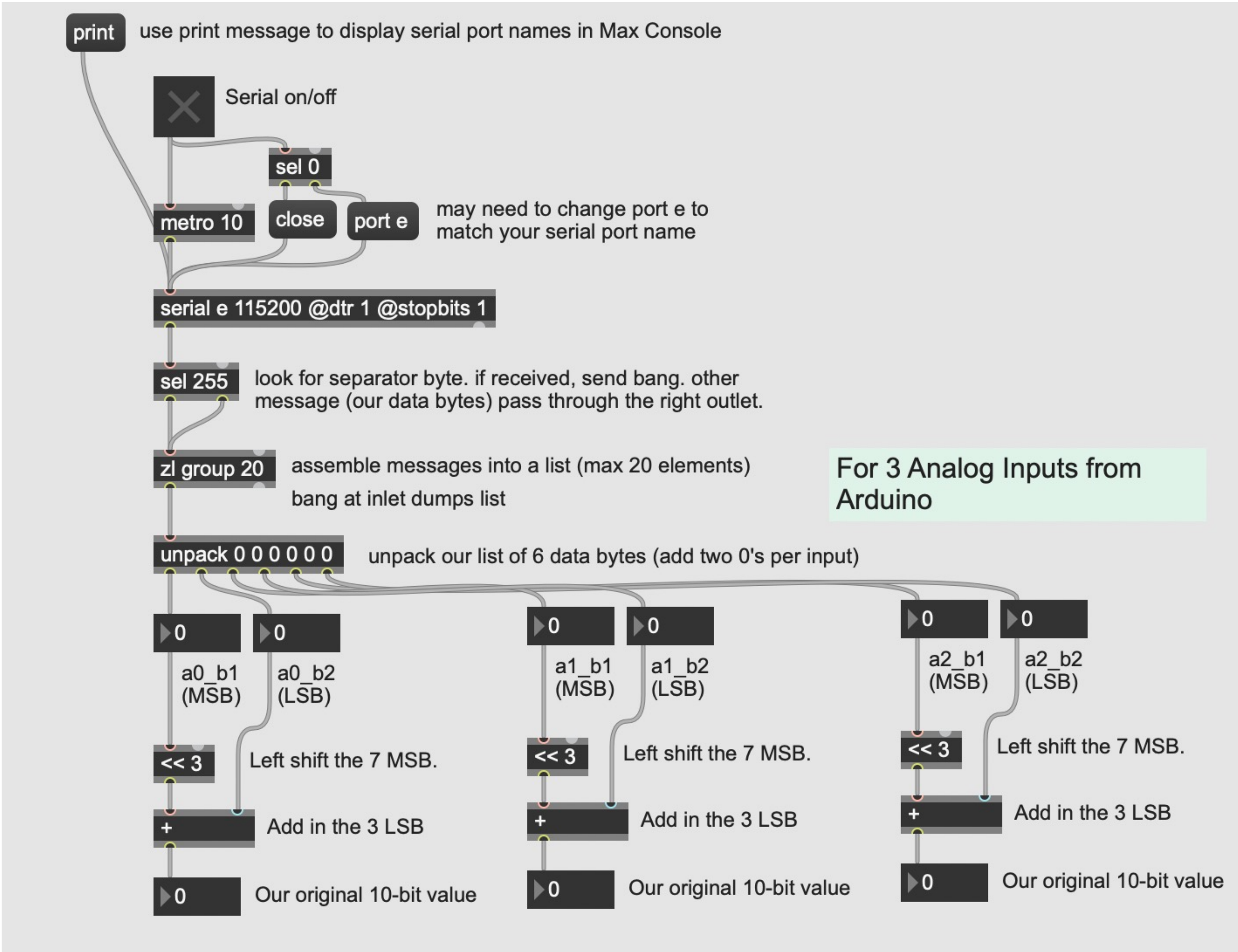
```
//WE assume this code is getting called in loop()

Serial.write(255);     //status byte

for (int i=0; i<n; i++) { //assumes n is defined
    int reading = analogRead(i); //range of 0-1023
    int b1 = reading >> 3;
    int b2 = reading & 7;

    Serial.write(b1);
    Serial.write(b2);
}
```

Updated Max Code for Parsing Our Serial Protocol



SOLUTION 2C (14 BIT VERSION)

- If we want to use 14-bit resolution, we can decompose our analog readings into two 7-bit bytes, as follows:

```
analogReadResolution(14);
int reading = analogRead(0); //range of 0-16383
int b1 = reading >> 7; //7 msb: range of 0-127
int b2 = reading & 127;  //7 lsb: range of 0-127
                          // 127 == 0b01111111
Serial.write(b1);
Serial.write(b2);
```

- At the receiving end, we will take the first byte and shift it back 7 places to the left, then add back in the second byte:

```
result = (b1<<7) + b2;
```

SOLUTION 2C (14 BIT VERSION)

- We can still use 255 as our “status byte” because our data bytes will still only be 7 bytes each, in the range of 0–127

```
Serial.write(255);      //status byte

for (int i=0; i<n; i++) { //assumes n is defined
    int reading = analogRead(i); //range of 0-1023
    int b1 = reading >> 7;
    int b2 = reading & 127;

    Serial.write(b1);
    Serial.write(b2);
}
```

14-bit Max Code for Parsing Our Serial Protocol