CPE201
Digital Design

By Benjamin Haas

Class 4: Floats, Text, Errors

University of Nevada, Reno

Clarification

- Thursday Lab
  - Makeups
  - Drop-in

Floating Point

- For numbers that are not whole
  - 230,879,521 = 2.30879521×10^{9} in decimal
  - Sign = positive, Exponent = 9, mantissa = 2.30879521
  - -152.49536 = -1.5249536×10^{2} in decimal
  - .00034789 = 3.4789×10^{-4} in decimal

Floating Point

- Sign, exponent, mantissa
- 1 bit, 8 bits , 23 bits = 32 bits (4 bytes)
- Sign (0 = positive, 1 = negative)
- Exponent (subtract 127)
  - Makes the range -126 to +128
- Mantissa always starts with non-zero number (1) so leave it out (subtract the first 1 bit)

Example

- Number = $(-1)^S(1+F)(2^{E-127})$
- Convert 56,231 to floating point
- 56,231 = $1101\ 1011\ 1010\ 0111_2$
- S = positive = 0
- F = $1.101\ 1011\ 1010\ 0111 \times 2^{15}$
- 1+F (E-127)
- E - 127 = 15 → E = 15 + 127 = 142 = 1000 1110
- Number = 0 1000 1110 101 1011 1010 0111 0000 0000
- Rearrange: 0100 0111 0101 1011 1010 0111 0000 0000
- Convert: 0x 4 7 5 B A 7 0 0
- In hex: 0x47 5B A7 00

Floating Point

- Pros:
  - Ability to represent non-while numbers
  - Large range of values
- Cons:
  - Math is a pain (and slow without an FPU)
  - Precision can be lost

ASCII

- 7 bit code for characters and symbols
  - Usually represent w/8bits and MSB=0
- For ‘simple’ data transfer

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Unicode

- Able to represent 2^{32} characters (4 Gchars)
  - 144,697 chars used in Unicode 14.0
- Typically encoded in UTF-8
  - Sends 1 to 4 bytes per character

Errors

- Verify that what you sent is what is received
- Creates data overhead
- Ensures data integrity
- Errors can happen over any transmission

Checksums

- Simple way to detect if there is an error
- Usually add a byte to make the message total equal zero

Example

- ASCII Encoded string “Hi There!”
- 0x48 0x69 0x20 0x54 0x68 0x65 0x72 0x65 0x21
- Add these together to get 0x2EA (truncated to 0xEA)
- Number to add to 0xEA to get 0x00 is 0x16)
- Or take 2’s complement of 0xEA (also get 0x16)
- Send original 9 bytes with 0x16 at the end
- Add together all 10 bytes, if you don’t get 0x00, there is an error in the message (usually throw it away and retry)

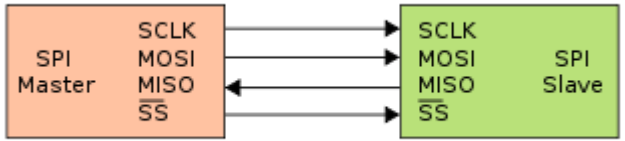
CRCs

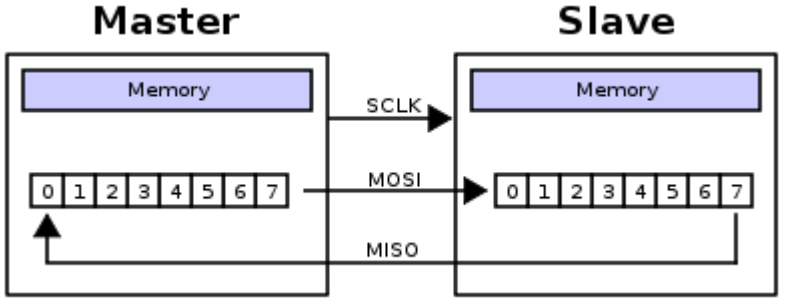
- Like a checksum, but more complicated
  - Better able to identify errors because the order of the data matters
  - More computationally intensive
  - More CRC bytes = better error detection capability at the cost of more transmission overhead

Transmission

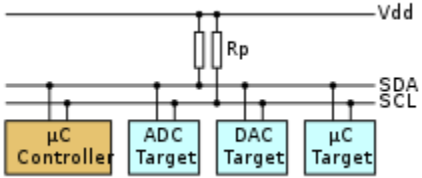
- UART (USART)

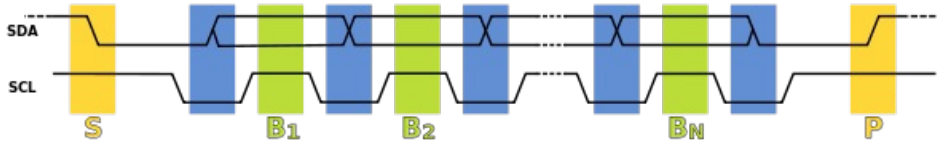
- SPI



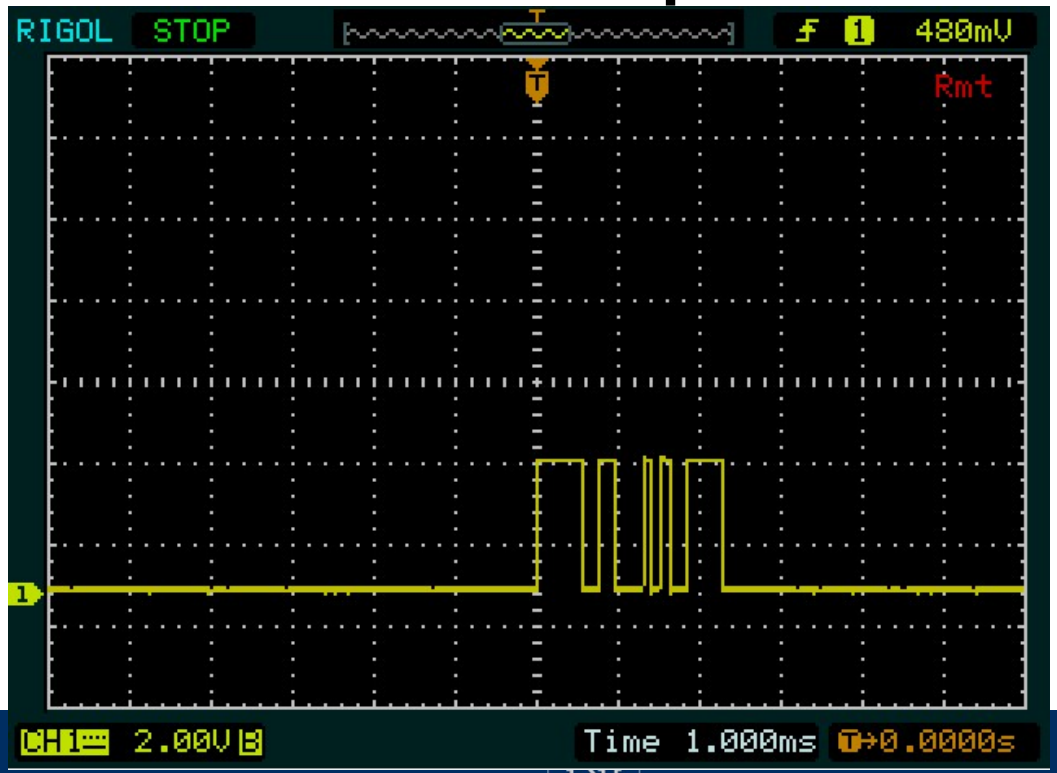


- I2C





Example



Explanation

- It is active low UART (3.3V transmission)
- 2 bytes transmitted
- Values “1010011110” and “1111100110”
- Removing START and STOP bits leaves
  - “01001111” and “11110011”
- Reversing the bits (UART send LSB first)
  - “11110010” and “11001111”
- Convert to hex 0xF2 and 0xCF
- Reversing byte order to make reading left to right (first byte of meesage sent first) makes the message 0xCF 0xF2
- ASCII encoding, message reads as 0 [CR]

Reading

- This lecture
  - Sections 2.6, 2.10-2.11
- Next lecture
  - Sections 3.1-3.3