Computing Systems
Binary representation

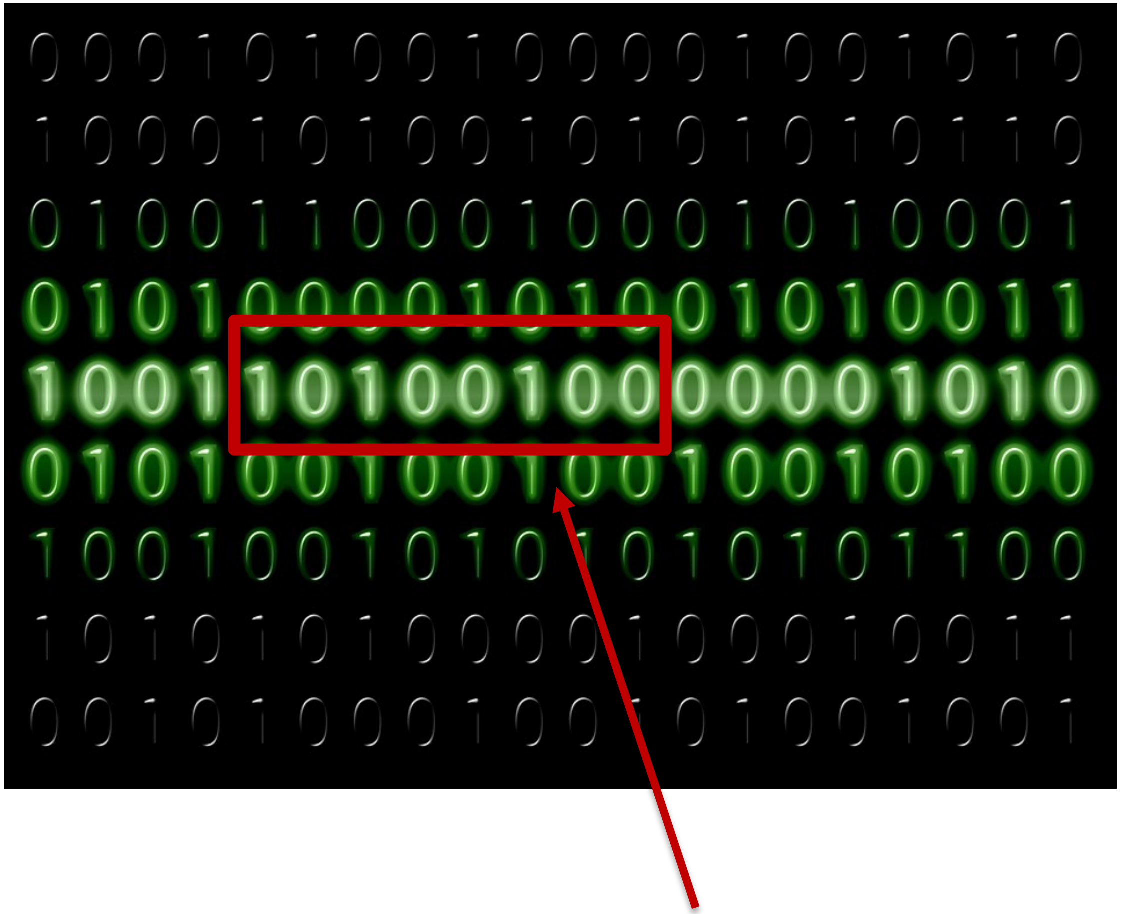
Andrea Masciadri, PhD
andrea.masciadri@polimi.it

Agenda

- Introduction
- Number representation
  - Binary, decimal, octal, hexadecimal
  - Sign representation
  - Floating point numbers
- Text representation

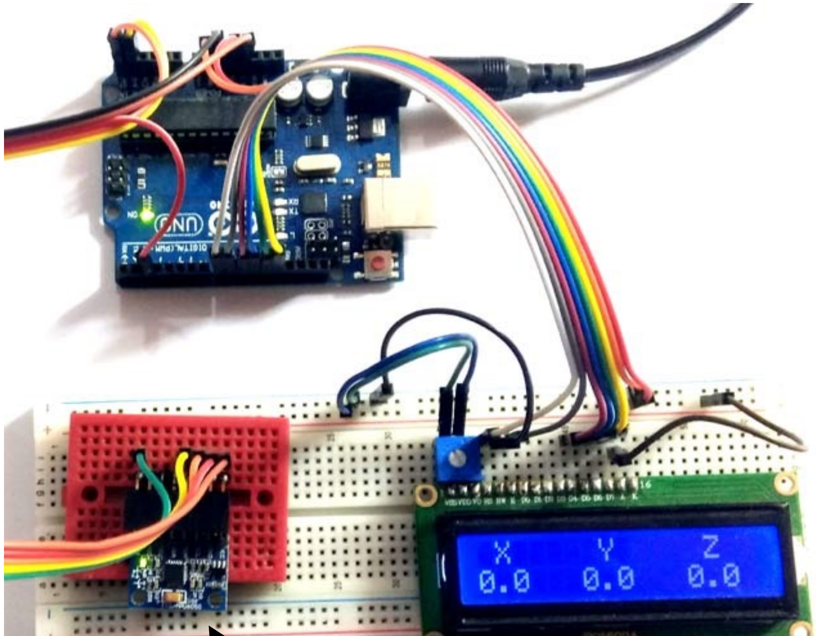
- Exercise

Introduction



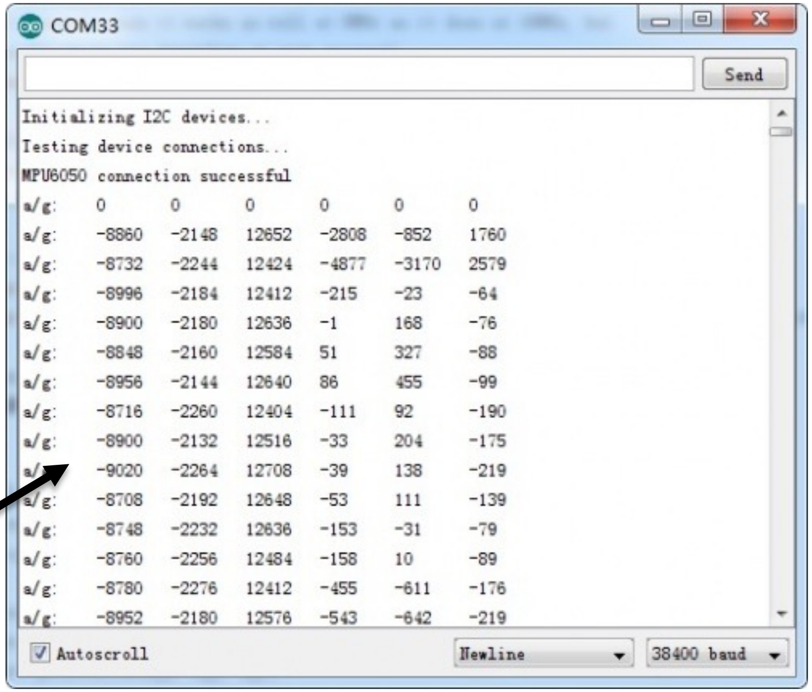
Who knows what this code means?

Introduction



MPU6050 Accelerometer

Who knows what these numbers mean?



Introduction

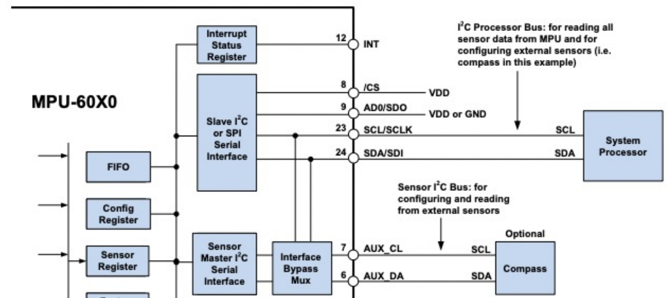
We need to open the MPU6050 datasheet to give meaning to that data.

7.13 MPU-60X0 Solution for 9-axis Sensor Fusion Using I²C Interface

In the figure below, the system processor is an I²C master to the MPU-60X0. In addition, the MPU-60X0 is an I²C master to the optional external compass sensor. The MPU-60X0 has limited capabilities as an I²C Master, and depends on the system processor to manage the initial configuration of any auxiliary sensors. The MPU-60X0 has an interface bypass multiplexer, which connects the system processor I²C bus pins 23 and 24 (SDA and SCL) directly to the auxiliary sensor I²C bus pins 6 and 7 (AUX_DA and AUX_CL).

Once the auxiliary sensors have been configured by the system processor, the interface bypass multiplexer should be disabled so that the MPU-60X0 auxiliary I²C master can take control of the sensor I²C bus and gather data from the auxiliary sensors.

For further information regarding I²C master control, please refer to Section 10.



InvenSense	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A Revision: 3.4 Release Date: 08/19/2013

6.2 Accelerometer Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		*g*	
	AFS_SEL=1		±4		*g*	
	AFS_SEL=2		±8		*g*	
	AFS_SEL=3		±16		*g*	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*	
	AFS_SEL=1		8,192		LSB/*g*	
	AFS_SEL=2		4,096		LSB/*g*	
	AFS_SEL=3		2,048		LSB/*g*	
Initial Calibration Tolerance			±3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			±2		%	
ZERO-G OUTPUT						
Initial Calibration Tolerance	X and Y axes		±50		mg	1
	Z axis		±80		mg	

Coding information is the basis of information technology

Binary representation

Computers store and process information in **binary**: a series of 1s and 0s.

Bit: smallest element, either 0 or 1

Byte: group of 8 bit

KiloByte: 2^{10} = 1024 bytes

MegaByte: 2^{20} bytes

GigaByte: 2^{30} bytes

TeraByte: 2^{40} bytes

Number representation

Decimal system (Base 10: using 10 digits from 0 to 9)

$$1234_{10} = 1*10^3 + 2*10^2 + 3*10^1 + 4*10^0$$

Binary system (Base 2: using 2 digits from 0 to 1)

$$1010_2 = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 10_{10}$$

Octal system (Base 8: using 8 digits from 0 to 7)

$$26_8 = 2*8^1 + 6*8^0 = 22_{10}$$

Hexadecimal system

(Base 16: using 16 digits from 0 to 9, A, B, C, D, E, F)

$$12B_{16} = 1*16^2 + 2*16^1 + 11*16^0 = 299_{10}$$

Binary representation

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
…	…
1111	15

How many numbers can be represented by a 4 digit binary number?

From 0000 to 1111
From 0 to 15
2^4 = 16 numbers

Sign representation

How can we represent a signed number?

$4_{10} = 0100_2$

$+4_{10} = ?$

$-4_{10} = ?$

Sign representation: signed magnitude

We can use one bit to represent the sign (0 = + , 1 = -)

$4_{10} = 0100_2$ unsigned binary

$+4_{10} = 0100_2$ signed magnitude

$-4_{10} = 1100_2$ signed magnitude

Sign representation: signed magnitude

Binary	Decimal
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

How many numbers can be represented by a 4 digit signed magnitude binary number?

From -7 to +7

2^4-1 = 15 numbers

we have two representation of the same number!

$0000_2 = 1000_2 = 0_{10}$

Sign representation: two's complement

The first bit is used to represent the sign (0 = + , 1 = -) but it is also used to represent the number

Positive numbers

$0100_2 = +4_{10}$

Simply the binary representation of the number

Negative numbers

1100_2

We have to compute the two's complement of the number:

From the less significative bit: leave the 0s and the first 1, reverse the rest

Sign representation: two's complement

Negative numbers

We have to compute the two's complement of the number:

From the less significative bit: leave the 0s and the first 1, reverse the rest

1111 Becomes 0001 = -1

1110 Becomes 0010 = -2

1101 Becomes 0011 = -3

1001 Becomes 0111 = -7

1000 Becomes 1000 = -8

Sign representation: signed magnitude

Binary	Decimal
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

How many numbers can be represented by a 4 digit signed magnitude binary number?

From -8 to +7
2^4 = 16 numbers

Decimal numbers

How can we represent decimal numbers?

$4_{10} = 0100_2$

$4.5_{10} = ?$

$4.724_{10} = ?$

Fixed point

We can decide how many bits to represent the integer and the fractional parts of the number (IIIIFFFF)

Examples:

00010110 fixed(8,3): 8bit number with 3bit fractional:
$00010.110_2 = 1*2^1 + 1*2^{-1} + 1*2^{-2} = 2 + 0.5 + 0.25 = 2.75$

00010110 fixed(8,5): 8bit number with 5bit fractional:
$000.10110_2 = 1*2^{-1} + 1*2^{-3} + 1*2^{-4} = 0.5 + 0.125 + 0.0625 = 0.6875$

Note that we have just shifted the numer with respect to the binary point.
Shifting a bit to the right by 1 position is equivalent to dividing the number by 2!

Fixed point 2’s complementation

Binary	Decimal(n)	FP(4,1)
0000	+0	0
0001	+1	0.5
0010	+2	1
0011	+3	1.5
0100	+4	2
0101	+5	2.5
0110	+6	3
0111	+7	3.5
1000	-8	-4
1001	-7	-3.5
1010	-6	-3
1011	-5	-2.5
1100	-4	-2
1101	-3	-1.5
1110	-2	-1
1111	-1	-0.5

Trade-off between:
Precision of the fractional part
Range of the integer part

Fixed point: operations

Be aware that overflows may occur.

Addition and subtraction:

Numbers with the same scaling factor. Just add or subtract.

Multiplication:

It suffices to multiply the two underlying integers, and assume that the scaling factor of the result is the product of their scaling factors.

Division:

One takes the integer quotient of their underlying integers, and assumes that the scaling factor is the quotient of their scaling factors. In general, the first division requires rounding and therefore the result is not exact. If the result is not exact, the error introduced by the rounding can be reduced or even eliminated by converting the dividend to a smaller scaling factor.

Floating point

Use scientific notation for binary numbers:

$$s\ 1.xxxx * 2^{yy}$$

s: sign $xxxx_2$: mantissa yy_2: exponent

Type	Minimum value	Maximum value
float	1.175494351 E - 38	3.402823466 E + 38
double	2.2250738585072014 E - 308	1.7976931348623158 E + 308

Text representation

Also characters are stored and processed in a binary representation.

ASCII: each character is represented by one byte (ok for English but insufficient for many languages)

UNICODE: standard that can be implemented by different character encodings (e.g. UTF8, UTF16, UTF32, UCS2, …) with different lengths.

ASCII table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Strings

Strings are just array of characters:

Using the ASCII character set:

- H = 72 = 01001000
- i = 105 = 01101001
- “Hi” = 01001000 01101001

Exercise

Get C0 and C1 values

8.11 Calibration Coefficients (COEF)

The Calibration Coefficients register contains the 2´s complement coefficients that are used to calculate the compensated pressure and temperature values.

Table 18 Calibration Coefficients

Coefficient	Addr.	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
c0	0x10	c0 [11:4]							
c0/c1	0x11	c0 [3:0]				c1 [11:8]			
c1	0x12	c1[7:0]							
c00	0x13	c00 [19:12]							
c00	0x14	c00 [11:4]							
c00/c10	0x15	c00 [3:0]				c10 [19:16]			
c10	0x16	c10 [15:8]							
c10	0x17	c10 [7:0]							
c01	0x18	c01 [15:8]							
c01	0x19	c01 [7:0]							
c11	0x1A	c11 [15:8]							
c11	0x1B	c11 [7:0]							
c20	0x1C	c20 [15:8]							
c20	0x1D	c20 [7:0]							
c21	0x1E	c21 [15:8]							
c21	0x1F	c21 [7:0]							
c30	0x20	c30 [15:8]							
c30	0x21	c30 [7:0]							

Exercise

init:

const int numReg = 18; // reading values from 0x10 a 0x20

```
....
  /*****************************************************/
  /*            Reading Coefficients           */
  /*****************************************************/
  Wire.beginTransmission(DPS368Address); // Begin transmission to the Sensor
  Wire.write(0x10); //register pointer set-up 0x10 -coefficient table-
  Wire.endTransmission();

  Wire.requestFrom(DPS368Address, numReg); // Request
  if(Wire.available()<= numReg) Wire.readBytes(COEFF, numReg); // Reads the data from the register

   /*  temperautra*/
   C0_HALF = (((unsigned int)COEFF[0] << 4) | (((unsigned int)COEFF[1] >> 4) & 0x0F)) >>1 ; //c0 is only used as
c0*0.5, so c0_half is calculated immediately
   C1 = (((unsigned int)COEFF[1] & 0x0F) << 8) | (unsigned int)COEFF[2]; //16bit!!!!

   if(C1>>11){
     C1 = C1 | 0xF000; /* if the value is negative, the sign must be fixed. The sign indication is in position 12. */
    }
```

References

- Chris Schmidt, Binary representation, https://slideplayer.com/slide/4901860/
- Hayden So, Introduction to Fixed Point Representation
- Wikipedia, Fixed point arithmetic https://en.wikipedia.org/wiki/Fixed-point_arithmetic

Questions?

andrea.masciadri@polimi.it