ÍNDICE DE ANEXOS

ANEXO A

Pruebas realizadas el 6 de diciembre de 2013, sin filtro digital, con las 4 etapas conectadas y presionando la plataforma de la celda de carga, el inamp se alimentó con +/- 6V.

vi	$b_{esperado}$ [dec]	$b_{esperado}$ [hex]	$b_{obtenido}$ [hex]	$b_{obtenido}$ [dec]	error [g]	error [%]
0.68	1361	550	530	1328	33	2%
0.72	1441	5A0	5af	1455	14	1%
0.76	1521	5F0	59b	1435	86	6%
0.76	1521	5F0	5cc	1484	37	2%
0.76	1521	5F0	5fd	1533	12	1%
0.76	1521	5F0	5f3	1523	2	0%
0.80	1601	640	655	1621	20	1%
0.80	1601	640	637	1591	10	1%
0.88	1761	6E0	6b6	1718	43	2%
0.88	1761	6E0	6ca	1738	23	1%
0.96	1921	780	77a	1914	7	0%
0.96	1921	780	797	1943	22	1%
1.00	2001	7D0	7be	1982	19	1%
1.12	2241	8C0	8cf	2255	14	1%
1.12	2241	8C0	8cf	2255	14	1%
1.12	2241	8C0	8f6	2294	53	2%
1.16	2321	910	927	2343	22	1%
1.20	2401	960	8f6	2294	107	5%
1.20	2401	960	8f6	2294	107	5%
1.24	2481	9B0	9d7	2519	38	2%
1.24	2481	9B0	9b0	2480	1	0%
1.28	2561	A01	99c	2460	101	4%
1.28	2561	A01	9e1	2529	32	1%
1.32	2641	A51	a60	2656	15	1%
1.36	2721	AA1	a6a	2666	55	2%
1.52	3041	BE1	bd3	3027	14	0%
1.56	3121	C31	c0d	3085	36	1%
1.60	3201	C81	c3e	3134	67	2%
1.60	3201	C81	c96	3222	21	1%
1.68	3361	D21	cf8	3320	41	1%
1.68	3361	D21	cd1	3281	80	2%
1.72	3441	D71	d6d	3437	4	0%
1.80	3601	E11	dcf	3535	66	2%
1.84	3681	E61	e1d	3613	68	2%
1.84	3681	E61	e61	3681	0	0%
1.88	3762	EB1	ea6	3750	12	0%
1.92	3842	F01	ed6	3798	44	1%
1.96	3922	F51	f86	3974	52	1%

1.96	3922	F51	f42	3906	16	0%
1.96	3922	F51	f1b	3867	55	1%
1.96	3922	F51	f55	3925	3	0%
2.00	4002	FA1	f42	3906	96	2%
2.00	4002	FA1	f5f	3935	67	2%
2.00	4002	FA1	f73	3955	47	1%
2.00	4002	FA1	f86	3974	28	1%
2.04	4082	FF1	fd4	4052	30	1%
2.16	4322	10E1	10c8	4296	26	1%
2.16	4322	10E1	1067	4199	123	3%
2.24	4482	1181	118c	4492	10	0%
2.36	4722	1271	123b	4667	55	1%
2.36	4722	1271	128a	4746	24	1%
2.40	4802	12C1	126c	4716	86	2%
2.44	4882	1311	1309	4873	9	0%
2.44	4882	1311	12ce	4814	68	1%
2.48	4962	1361	131c	4892	70	1%
2.48	4962	1361	1357	4951	11	0%
2.52	5042	13B2	134d	4941	101	2%
2.52	5042	13B2	1374	4980	62	1%
2.60	5202	1452	142e	5166	36	1%
2.60	5202	1452	13fd	5117	85	2%
2.60	5202	1452	141a	5146	56	1%
2.64	5282	14A2	145e	5214	68	1%
2.68	5362	14F2	14f1	5361	1	0%
2.72	5442	1542	1504	5380	62	1%
2.76	5522	1592	14fb	5371	151	3%
2.80	5602	15E2	15be	5566	36	1%
2.84	5682	1632	1647	5703	21	0%
2.84	5682	1632	15ef	5615	67	1%
2.88	5762	1682	1616	5654	108	2%
2.88	5762	1682	1647	5703	59	1%
2.92	5842	16D2	16a8	5800	42	1%
2.96	5922	1722	1700	5888	34	1%
3.00	6002	1772	16cf	5839	163	3%
3.00	6002	1772	16cf	5839	163	3%
3.16	6323	18B2	18ae	6318	5	0%
3.20	6403	1902	18c1	6337	66	1%
3.32	6643	19F2	19bf	6591	52	1%
3.36	6723	1A42	19e6	6630	93	1%
3.40	6803	1A92	1a65	6757	46	1%
3.40	6803	1A92	1a52	6738	65	1%
3.44	6883	1AE2	1aa0	6816	67	1%
3.52	7043	1B82	1ad1	6865	178	3%
3.52	7043	1B82	1b02	6914	129	2%

3.52	7043	1B82	1b6d	7021	22	0%
3.64	7283	1C72	1bff	7167	116	2%
3.68	7363	1CC2	1cb0	7344	19	0%
3.72	7443	1D12	1c92	7314	129	2%
3.72	7443	1D12	1cd6	7382	61	1%
3.80	7603	1DB3	1d73	7539	64	1%
3.96	7923	1EF3	1eb5	7861	62	1%
3.96	7923	1EF3	1eab	7851	72	1%
4.08	8163	1FE3	1f95	8085	78	1%
4.08	8163	1FE3	1f82	8066	97	1%
4.36	8723	2213	217e	8574	149	2%
4.44	8884	22B3	227c	8828	56	1%
4.56	9124	23A3	2397	9111	13	0%
4.60	9204	23F3	2383	9091	113	1%
4.76	9524	2533	2531	9521	3	0%
5.08	10164	27B4	2706	9990	174	2%
5.08	10164	27B4	2706	9990	174	2%

Pruebas realizadas el 6 de diciembre de 2013, usando filtro digital, con las 4 etapas conectadas y presionando la plataforma de la celda de carga, el inamp se alimentó con +/- 6V.

vi	$b_{esperado}$ [dec]	$b_{esperado}$ [hex]	$b_{obtenido}$ [hex]	$b_{obtenido}$ [dec]	error [g]	error [%]
0.48	960	3C0	3da	986	26	3%
0.56	1120	460	46c	1132	12	1%
0.76	1521	5F0	5f3	1523	2	0%
0.92	1841	730	70E	1806	35	2%
0.92	1841	730	718	1816	25	1%
0.92	1841	730	749	1865	24	1%
0.96	1921	780	753	1875	46	2%
0.96	1921	780	75c	1884	37	2%
0.96	1921	780	766	1894	27	1%
1.04	2081	820	829	2089	8	0%
1.04	2081	820	847	2119	38	2%
1.08	2161	870	850	2128	33	2%
1.12	2241	8C0	881	2177	64	3%
1.16	2321	910	90A	2314	7	0%
1.16	2321	910	914	2324	3	0%
1.16	2321	910	914	2324	3	0%
1.16	2321	910	945	2373	52	2%
1.20	2401	960	962	2402	1	0%
1.48	2961	B91	b98	2968	7	0%
1.52	3041	BE1	ba2	2978	63	2%

1.56	3121	C31	c48	3144	23	1%
1.56	3121	C31	c35	3125	4	0%
1.60	3201	C81	c5c	3164	37	1%
1.60	3201	C81	c52	3154	47	1%
1.64	3281	CD1	cd1	3281	0	0%
1.64	3281	CD1	CDB	3291	10	0%
1.64	3281	CD1	CC7	3271	10	0%
1.64	3281	CD1	cdb	3291	10	0%
1.68	3361	D21	d3c	3388	27	1%
1.72	3441	D71	D6D	3437	4	0%
1.72	3441	D71	d81	3457	16	0%
1.72	3441	D71	d50	3408	33	1%
1.72	3441	D71	d1f	3359	82	2%
1.72	3441	D71	d81	3457	16	0%
1.80	3601	E11	e09	3593	8	0%
1.84	3681	E61	E7E	3710	29	1%
1.96	3922	F51	f7c	3964	42	1%
1.96	3922	F51	f73	3955	33	1%
1.96	3922	F51	f9a	3994	72	2%
2.12	4242	1091	1098	4248	6	0%
2.12	4242	1091	1084	4228	14	0%
2.12	4242	1091	107a	4218	24	1%
2.20	4402	1131	10e6	4326	76	2%
2.24	4482	1181	119f	4511	29	1%
2.28	4562	11D1	1195	4501	61	1%
2.32	4642	1221	11f7	4599	43	1%
2.32	4642	1221	120b	4619	23	0%
2.36	4722	1271	128a	4746	24	1%
2.36	4722	1271	1276	4726	4	0%
2.36	4722	1271	12e1	4833	111	2%
2.44	4882	1311	1312	4882	0	0%
2.48	4962	1361	136a	4970	8	0%
2.48	4962	1361	1357	4951	11	0%
2.48	4962	1361	134d	4941	21	0%
2.48	4962	1361	1374	4980	18	0%
2.60	5202	1452	144b	5195	7	0%
2.64	5282	14A2	147c	5244	38	1%
2.76	5522	1592	1583	5507	15	0%
2.80	5602	15E2	15aa	5546	56	1%
2.84	5682	1632	1616	5654	28	1%
2.88	5762	1682	1633	5683	79	1%
2.88	5762	1682	1664	5732	30	1%
2.88	5762	1682	1647	5703	59	1%
2.92	5842	16D2	16BC	5820	22	0%
2.92	5842	16D2	16d9	5849	7	0%

2.92	5842	16D2	1695	5781	61	1%
2.96	5922	1722	16cf	5839	83	1%
3.00	6002	1772	173b	5947	55	1%
3.08	6162	1812	17eb	6123	39	1%
3.08	6162	1812	17A6	6054	108	2%
3.16	6323	18B2	189a	6298	25	0%
3.24	6483	1952	1923	6435	48	1%
3.28	6563	19A2	197B	6523	40	1%
3.32	6643	19F2	19bf	6591	52	1%
3.36	6723	1A42	19f0	6640	83	1%
3.40	6803	1A92	1a65	6757	46	1%
3.44	6883	1AE2	1A65	6757	126	2%
3.68	7363	1CC2	1c3a	7226	137	2%
3.68	7363	1CC2	1C57	7255	108	1%
3.68	7363	1CC2	1c44	7236	127	2%
3.76	7523	1D63	1d38	7480	43	1%
3.80	7603	1DB3	1d73	7539	64	1%
3.84	7683	1E03	1da3	7587	96	1%
3.84	7683	1E03	1de8	7656	27	0%
3.88	7763	1E53	1dfb	7675	88	1%
3.92	7843	1EA3	1eb5	7861	18	0%
3.92	7843	1EA3	1eb5	7861	18	0%
3.92	7843	1EA3	1ed2	7890	47	1%
3.92	7843	1EA3	1e67	7783	60	1%
3.92	7843	1EA3	1e53	7763	80	1%
4.08	8163	1FE3	1f9f	8095	68	1%
4.20	8403	20D3	2080	8320	83	1%
4.28	8563	2173	2126	8486	77	1%
4.32	8643	21C3	21a5	8613	30	0%
4.40	8804	2263	221a	8730	74	1%
4.48	8964	2303	22c0	8896	68	1%
4.48	8964	2303	22dd	8925	39	0%
4.56	9124	23A3	232b	9003	121	1%
4.56	9124	23A3	2370	9072	52	1%
4.88	9764	2623	261b	9755	9	0%

ANEXO B

Código del programa que va en el microcontrolador. Se utilizó el lenguaje ensamblador del ATmega88.

```
.include "C:\VMLAB\include\m88def.inc"

        .equ    NumBytes        =       2
        .equ    zone_limit      =       40
        .equ    H_inv           =       625
        .equ    prohibido       =       $C3

; Define here the variables
.dseg
        .org    $100
        char_buff       :       .byte   16
        address         :       .byte   2
        mea_val         :       .byte   2
        reference       :       .byte   2
        final_val       :       .byte   2
        med_en_pro      :       .byte   2

        .org    $120
         num1           :       .byte NumBytes
        num2            :       .byte NumBytes
        result          :       .byte   2*NumBytes
        int_mul         :       .byte 2*NumBytes*NumBytes
        DSP_buffer      :       .byte   16
        sum_11bits1     :       .byte   2
        sum_11bits2     :       .byte   2
        sum_11bits3     :       .byte   2
        sum_11bits4     :       .byte   2
        sum_12bits1     :       .byte   2
        sum_12bits2     :       .byte   2
        sum_13bits      :       .byte   2

.cseg
        .org    0

        .def    DATO            =       R25
        .def    flags           =       R24
        .def    char_counter    =       R23
        .def    comm_index      =       R22

        .def    mcarry          =       R19
        .def    front_zeros     =       R20
        .def    back_zeros      =       R21

        .equ    Fosc            =       1000000
        .equ    bps             =       4800
        .equ    num_comm        =       5
```

```
; Define here Reset and interrupt vectors, if any
reset:
  rjmp start
  reti    ; Addr $01
  reti    ; Addr $02
  reti    ; Addr $03
  reti    ; Addr $04
  reti    ; Addr $05
  reti    ; Addr $06
  reti    ; Addr $07
  reti    ; Addr $08
  reti    ; Addr $09
  reti    ; Addr $0A
  reti    ; Addr $0B
  reti    ; Addr $0C
  reti    ; Addr $0D
  reti    ; Addr $0E
  reti    ; Addr $0F
  reti    ; Addr $10
  reti    ; Addr $11
  rjmp    RX_complete          ;Addr $12
  reti    ; Addr $13
  reti    ; Addr $14
  rjmp  ADC_conv_complete     ;Addr $15
  reti    ; Addr $16
  reti    ; Addr $17
  reti    ; Addr $18
  reti    ; Addr $19

; Program starts here after Reset
start:

        ldi     R16,    high(RAMEND)
        out     SPH,    R16
        ldi     R16,    low(RAMEND)
        out     SPL,    R16

        ;---------- initial conditions  ----------

        ;ASIGNAMOS UNA DIRECCIÓN INICIAL AL SISTEMA
        ldi     XH,     high(address)
        ldi     XL,     low(address)
        ldi     R16,    '3'
        st      X+,     R16
        ldi     R16,    '2'
        st      X+,     R16

        ;FACTOR MULTIPLICATIVO PARA CONVERSIÓN BITS-->GRAMOS
        ldi     R16,            low(H_inv)
        sts     num2,           R16
        ldi     R16,            high(H_inv)
```

```
sts     num2+1,         R16

;FLAGS INICIALES
ldi     comm_index,     0
ldi     flags,          0
ldi     char_counter,   0

;VALORES EN RAM INICIALES
ldi     YH,     high(reference)
ldi     YL,     low(reference)
ldi     R16,    $00
st      Y+,     R16
st      Y+,     R16
sts     med_en_pro    ,         R16
sts     med_en_pro+1,           R16
sts     final_val     ,         R16
sts     final_val+1   ,         R16
sts     result        ,         R16
sts     result+1      ,         R16
;-----------------------------------------

rcall   config_USART

rcall   config_ADC

ldi     YH,     high(DSP_buffer)
ldi     YL,     low(DSP_buffer)

process_reset:

ldi     XH,             high(char_buff)
ldi     XL,             low(char_buff)
ldi     char_counter,   0
andi    flags,          0b01010100
sei

process:

        mov     R16,            flags
        andi    R16,            0b00000001
        cpi     R16,            0b00000001
        breq    there_was_a_59
        rjmp    process

there_was_a_59:

cli
rcall   identify_command

;--------- execute a sub-routine according to the command index --------
comm_1:
cpi     comm_index,     1
```

```
        brne    check_selected
        rcall   exe_s

        check_selected:
        mov     R16,    flags
        andi    R16,    0b00000100
        cpi     R16,    0b00000100
        brne    comm_5

        comm_0:
        cpi     comm_index,   0
        brne    comm_2
        rcall   exe_inval

        comm_2:
        cpi     comm_index,   2
        brne    comm_3
        rcall   exe_adr

        comm_3:
        cpi     comm_index,   3
        brne    comm_4
        rcall   exe_tar

        comm_4:
        cpi     comm_index,   4
        brne    comm_5
        rcall   exe_tas

        comm_5:
        cpi     comm_index,   5
        brne    comm_6
        rcall   exe_avg

        comm_6:
        rjmp    process_reset


forever:
        rjmp forever
;*****************************************************
;*****************************************************
;********** INTERRUPTION SERVICE ROUTINE ************
;*****************************************************
;*****************************************************

RX_complete:

        push    R16
        in      R16,    SREG
        push    R16
```

```
        lds     R16,    UDR0
        st      X+,     R16

        cpi     R16,    59
        breq    RX_OK

        inc     char_counter
        cpi     char_counter,   16
        brne    RX_end

        ldi     XH,     high(char_buff)
        ldi     XL,     low(char_buff)
        rjmp    RX_end

        RX_OK:
        ori     flags,  0b00000001

        RX_end:

        pop     R16
        out     SREG,   R16
        pop     R16

reti

ADC_conv_complete:

        push    R16
        in      R16,            SREG
        push    R16
        push    R17

        lds     R17,            ADCL
        lds     R16,            ADCH

        sts     mea_val,        R17
        sts     mea_val+1,      R16

        ;¿mayor que el zone_limit?
        cpi     R16,            0
        brne    zone_1_jump
        cpi     R17,            zone_limit ;
        brsh    zone_1_jump

        zone_0:
        mov     R16,    flags
        andi    R16,    0b00100000
        cpi     R16,    0b00100000
        brne    zone_0_end

        mov     R16,    flags
```

```
andi    R16,    0b00010000
cpi     R16,    0b00010000
breq    TAS_ON

        TAS_OFF:
        lds     R16,            med_en_pro
        sts     final_val,      R16
        lds     R16,            med_en_pro+1
        sts     final_val+1,    R16
  rjmp  zone_0_process

        TAS_ON:
        lds     R16,            med_en_pro
        lds     R17,            reference
        clc
        sbc     R16,            R17
        sts     final_val,      R16
        lds     R16,            med_en_pro+1
        lds     R17,            reference+1
        sbc     R16,            R17
        sts     final_val+1,    R16

zone_0_process:
;NOW WE PROCESS THE INFO : WEIGHT-->ASCII
lds     R16,            final_val
sts     num1,           R16
lds     R16,            final_val+1
sts     num1+1,         R16

rcall   mult_uriol

rjmp    jump1
zone_1_jump:
rjmp    zone_1
jump1:

ldi     R20,    NumBytes*2
ldi     R21,    6
ldi     XH,     high(result)
ldi     XL,     low(result)
rcall   shift_RAM_right

;SI ES C3 , CANCELAR !!
lds     R16,    result+1
cpi     R16,    high(prohibido)
breq    maybe_prohibido
rjmp    no_prohibido

maybe_prohibido:
lds     R16,    result
cpi     R16,    low(prohibido)
breq    zone_0_end ; es el límite, por lo tanto no cuenta
```

```
no_prohibido:

lds     DATO,  result
rcall   tx
lds     DATO,  result+1
rcall   tx
rcall   enter

zone_0_end:
andi    flags,   0b11011111

rjmp    ADC_conv_complete_end

zone_1:
;¿el momento anterior estaba en la zona 0?
mov     R16,     flags
andi    R16,     0b00100000
cpi     R16,     0b00000000
brne    zone_1_skip

;en caso de haber estado en la zona 0, borrar la última med-en-pro
zone_1_clear_last_val:
ldi     R16,                0
sts     med_en_pro    ,     R16
sts     med_en_pro+1,       R16

zone_1_skip:
;flag(zone)=1
ori     flags,   0b00100000

mov     R16,     flags
andi    R16,     0b01000000
breq    NO_DSP

;SI DSP
cpi     R18,     8
breq    average

;keep storing samples, Y must remain constant in every dash
lds     R16,     mea_val
st      Y+,      R16
lds     R16,     mea_val+1
st      Y+,      R16
inc     R18
rjmp    ADC_conv_complete_end

average:
clr     R18
ldi     YH,      high(DSP_buffer)
ldi     YL,      low(DSP_buffer)
rcall   sample_average
```

```
        NO_DSP:
        ;comparar mea-val y med-en-pro
        ;empezamos por los bytes más significativos
        lds     R16,            mea_val+1
        lds     R17,            med_en_pro+1
        cp      R16,            R17
        brlo    zone_1_end      ; si mea-val es menor que med-en-pro, pasa de largo

        ;puede que sea igual o mayor
        cp      R16,            R17
        breq    zone_1_compare  ; si son iguales, habría que evaluar los bytes menos
significativos
        rjmp    zone_1_refresh  ; si no es igual, entonces es mayor, entonces actualizamos
med-en-pro

        zone_1_compare:
        lds     R16,            mea_val
        lds     R17,            med_en_pro
        cp      R17,            R16
        brsh    ADC_conv_complete_end       ; si med-en-pro es igual o mayor a mea-val, no
actualizar nada

        zone_1_refresh:
        lds     R16,            mea_val
        sts     med_en_pro,     R16
        lds     R16,                    mea_val+1
        sts     med_en_pro+1,           R16

        zone_1_end:

        ADC_conv_complete_end:

        pop     R17
        pop     R16
        out     SREG,   R16
        pop     R16

reti

;*****************************************************
;*****************************************************
;******************** SUB-ROUTINE ********************
;*****************************************************
;*****************************************************

sample_average:

        push    XH
        push    XL
        push    YH
        push    YL
```

```
push    ZH
push    ZL
push    R18
push    R19

ldi     XH,     high(DSP_buffer)
ldi     XL,     low(DSP_buffer)
ldi     YH,     high(DSP_buffer+2)
ldi     YL,     low(DSP_buffer+2)
ldi     ZH,     high(sum_11bits1)
ldi     ZL,     low(sum_11bits1)

rcall   add_RAM

ldi     XH,     high(DSP_buffer+4)
ldi     XL,     low(DSP_buffer+4)
ldi     YH,     high(DSP_buffer+6)
ldi     YL,     low(DSP_buffer+6)
ldi     ZH,     high(sum_11bits2)
ldi     ZL,     low(sum_11bits2)

rcall   add_RAM

ldi     XH,     high(DSP_buffer+8)
ldi     XL,     low(DSP_buffer+8)
ldi     YH,     high(DSP_buffer+10)
ldi     YL,     low(DSP_buffer+10)
ldi     ZH,     high(sum_11bits3)
ldi     ZL,     low(sum_11bits3)

rcall   add_RAM

ldi     XH,     high(DSP_buffer+12)
ldi     XL,     low(DSP_buffer+12)
ldi     YH,     high(DSP_buffer+14)
ldi     YL,     low(DSP_buffer+14)
ldi     ZH,     high(sum_11bits4)
ldi     ZL,     low(sum_11bits4)

rcall   add_RAM

ldi     XH,     high(sum_11bits1)
ldi     XL,     low(sum_11bits1)
ldi     YH,     high(sum_11bits2)
ldi     YL,     low(sum_11bits2)
ldi     ZH,     high(sum_12bits1)
ldi     ZL,     low(sum_12bits1)

rcall   add_RAM

ldi     XH,     high(sum_11bits3)
ldi     XL,     low(sum_11bits3)
```

```
        ldi     YH,     high(sum_11bits4)
        ldi     YL,     low(sum_11bits4)
        ldi     ZH,     high(sum_12bits2)
        ldi     ZL,     low(sum_12bits2)

        rcall   add_RAM

        ldi     XH,     high(sum_12bits1)
        ldi     XL,     low(sum_12bits1)
        ldi     YH,     high(sum_12bits2)
        ldi     YL,     low(sum_12bits2)
        ldi     ZH,     high(sum_13bits)
        ldi     ZL,     low(sum_13bits)

        rcall   add_RAM

        ldi     R20,    NumBytes
        ldi     R21,    3
        ldi     XH,     high(sum_13bits)
        ldi     XL,     low(sum_13bits)
        rcall   shift_RAM_right

        lds     R16,            sum_13bits
        sts     mea_val,        R16
        lds     R16,            sum_13bits+1
        sts     mea_val+1,      R16

        pop     R19
        pop     R18
        pop     ZL
        pop     ZH
        pop     YL
        pop     YH
        pop     XL
        pop     XH

ret

;  add two numbers stored in the RAM
;       the result will be stored in the RAM too
;       both numbers have the same size (in bytes)
;       it takes 0.4 ms at 1MHz clock speed
;
;       INPUTs:
;                               X               :       the RAM direction of the first number
;                               Y               :       the RAM direction of the second
number
;                               Z               :       the RAM direction of the result
;
;       OUTPUTs:
;                               a space in the RAM with the result
add_RAM:
```

```
        push    R16
        push    R17
        push    R18
        push    R19

        ldi     R18,    0
        ldi     R19,    NumBytes

        add_RAM_loop:

                cpi     R19,    0
                breq    add_RAM_last
                dec     R19

                ld      R16,    X+
                ld      R17,    Y+

                add     R16,    R18     ;add the carry from the previous sum
                add     R16,    R17   ;add the numbers

                st      Z+,     R16
                clr     R18

                brcc    no_carry
                inc     R18
                rjmp    add_RAM_loop

                no_carry:
                ldi     R18,    0
                rjmp    add_RAM_loop

                add_RAM_last:
                st      Z+,     R18

                pop     R19
                pop     R18
                pop     R17
                pop     R16

ret


exe_inval:

        push    R16

        mov     R16,    flags
        andi    R16,    0b00000100
        cpi     R16,    0b00000100
        brne    exe_inval_end
        ldi     DATO,   '?'
```

```
        rcall   TX
        rcall   enter

        exe_inval_end:

        pop     R16

ret

exe_s:

        push    YH
        push    YL
        ldi     XH,     high(char_buff)
        ldi     XL,     low(char_buff)
        ldi     YH,     high(address)
        ldi     YL,     low(address)
        adiw    XH:XL,  1

        mov     R18,    char_counter
        dec     R18

        exe_s_loop:
                ld      R16,    X+
                ld      R17,    Y+

                cpi     R17,    '0'
                breq    exe_s_maybe_1_char
                rjmp    exe_s_2_chars

    exe_s_maybe_1_char:
    cpi  R18,   1
    breq        exe_s_indeed_1_char
    rjmp        exe_s_KO

                exe_s_indeed_1_char:
                ld      R17,    Y
                cp      R16,    R17
                breq    exe_s_OK
                rjmp    exe_s_KO

                exe_s_2_chars:
                cp      R16,    R17
                brne    exe_s_KO
                dec     R18
                cpi     R18,    0
                breq    exe_s_OK
                ld      R16,    X+
                ld      R17,    Y+
                rjmp    exe_s_2_chars

        exe_s_KO:
```

```
        ;BANDERA QUE INDICA QUE EL SISTEMA HA SIDO DESACTIVADO
        andi    flags,          0b11111011
        ;APAGAR EL ADC
        ldi     R16,            0b01101011
        sts     ADCSRA,         R16
        rjmp    exe_s_end

        exe_s_OK:
        ;BANDERA QUE INDICA QUE EL SISTEMA HA SIDO ACTIVADO
        ori     flags,          0b00000100
        ;ENCENDER EL ADC
        ldi     R16,            0b11101011
        sts     ADCSRA,         R16
        rjmp    exe_s_end

        exe_s_end:

        pop     YL
        pop     YH

ret

exe_adr:

  push  YH
  push  YL

        ldi     XH,     high(char_buff)
        ldi     XL,     low(char_buff)
        ldi     YH,     high(address)
        ldi     YL,     low(address)
        adiw    XH:XL,  3

        ld      R16,    X+

        cpi     char_counter,   4
        brne    adr_5_char
        cpi     R16,            '?'
        breq    query

        ldi     R17,    '0'
        st      Y+,     R17
        st      Y,      R16
        rjmp    exe_adr_end

        query:
        ld      DATO,   Y+
        rcall   TX
        ld      DATO,   Y
        rcall   TX
        rcall   enter
        rjmp    exe_adr_end
```

```
        adr_5_char:
        cpi     char_counter,   5
        brne    exe_adr_end

        rcall   validar_num_ascii
        mov     R18,    flags
        andi    R18,    0b00001000
        cpi     R18,    0b00001000
        brne    exe_adr_end
        ld      R16,    X
        rcall   validar_num_ascii
        mov     R18,    flags
        andi    R18,    0b00001000
        cpi     R18,    0b00001000
        brne    exe_adr_end

        ldi     XH,     high(char_buff)
        ldi     XL,     low(char_buff)
        adiw    XH:XL,  3

        ld      R16,    X+
        st      Y+,     R16
        ld      R16,    X
        st      Y,      R16

        ldi     DATO,   '0'
        rcall   TX
        rcall   enter

        exe_adr_end:

ret


exe_tar:

        cpi     char_counter,   3
        brne    TAR_comm_inval

        nop
        ldi     XH,     high(mea_val)
        ldi     XL,     low(mea_val)
        ldi     YH,     high(reference)
        ldi     YL,     low(reference)

        ld      R16,    X+
        st      Y+,     R16
        ld      R16,    X+
        st      Y+,     R16
        rjmp    TAR_end
```

```
        TAR_comm_inval:
        ldi     DATO,  '?'
        rcall   TX
        rcall   enter
        rjmp    exe_TAS_end

        TAR_end:

        pop     YL
        pop     YH

ret

exe_tas:

        cpi     char_counter,  4
        brne    TAS_comm_inval

        ldi     XH,     high(char_buff)
        ldi     XL,     low(char_buff)
        adiw    XH:XL,  3

        ld      R16,    X

        cpi     R16,    '?'
        breq    TAS_query

        TAS_comm:
        cpi     R16,    '1'
        brne    TAS_is_it_zero

        TAS_comm_1:
        ori     flags,  0b00010000
        rjmp    TAS_comm_end

        TAS_is_it_zero:
        cpi     R16,    '0'
        brne    TAS_comm_inval

        TAS_comm_0:
        andi    flags,0b11101111
        rjmp    TAS_comm_end

        TAS_comm_inval:
        ldi     DATO,  '?'
        rcall   TX
        rcall   enter
        rjmp    exe_TAS_end

        TAS_query:
        mov     R17,    flags
        andi    R17,    0b00010000
```

```
        cpi     R17,    0b00010000
        breq    TAS_query_1

        TAS_query_0:
        ldi     DATO,   '0'
        rcall   TX
        rcall   enter
        rjmp    exe_TAS_end

        TAS_query_1:
        ldi     DATO,   '1'
        rcall   TX
        rcall   enter
        rjmp    exe_TAS_end

        TAS_comm_end:
        ldi     DATO,   '0'
        rcall   TX
        rcall   enter

        exe_TAS_end:

ret

exe_avg:

        cpi     char_counter,   4
        brne    AVG_comm_inval

        ldi     XH,     high(char_buff)
        ldi     XL,     low(char_buff)
        adiw    XH:XL,3

        ld      R16,    X

        cpi     R16,    '?'
        breq    AVG_query

        AVG_comm:
        cpi     R16,    '1'
        brne    AVG_is_it_zero

        AVG_comm_1:
        ori     flags,  0b01000000
        rjmp    AVG_comm_end

        AVG_is_it_zero:
        cpi     R16,    '0'
        brne    AVG_comm_inval

        AVG_comm_0:
        andi    flags,  0b10111111
```

```
        rjmp    AVG_comm_end

        AVG_comm_inval:
        ldi     DATO,  '?'
        rcall   TX
        rcall   enter
        rjmp    exe_AVG_end

        AVG_query:
        mov     R17,    flags
        andi    R17,    0b01000000
        cpi     R17,    0b01000000
        breq    AVG_query_1

        AVG_query_0:
        ldi     DATO,  '0'
        rcall   TX
        rcall   enter
        rjmp    exe_AVG_end

        AVG_query_1:
        ldi     DATO,  '1'
        rcall   TX
        rcall   enter
        rjmp    exe_AVG_end

        AVG_comm_end:
        ldi     DATO,  '0'
        rcall   TX
        rcall   enter

        exe_AVG_end:

ret

identify_command:

        push    R16
        push    R17

        ldi     XH,     high(char_buff)
        ldi     XL,     low(char_buff)
        ldi     ZH,     high(ADR*2)
        ldi     ZL,     low(ADR*2)
        ldi     R17,    num_comm
        ldi     comm_index,   0

        cpi     char_counter,  0
        breq    invalid_comm

        cpi     char_counter,  1
        breq    invalid_comm
```

```
        cpi     char_counter,   2
        breq    ident_S
        rjmp    ident_3_char

        ident_S:
        ld      R16,    X
        cpi     R16,    'S'
        brne    not_S
        ldi     comm_index,   1
        rjmp    identify_comm_end

        ident_3_char:
        cpi     char_counter,   3
        breq    ident_S

        not_S:
        inc     comm_index

        identify_comm_loop:

                inc     comm_index
                rcall   compare_char_by_char
                dec     R17

                mov     R16,    flags
                andi    R16,    0b00000010
                cpi     R16,    0b00000010
                breq    identify_comm_end

                cpi     R17,    0
                breq    identify_comm_end
                rjmp    identify_comm_loop

        identify_comm_end:
        ;mov    R16,    flags
        ;andi   R16,    0b00000010
        cpi     R16,    0b00000000
        breq    invalid_comm
        rjmp    ident_comm_end

        invalid_comm:
        ldi     comm_index,   0 ; comando inválido

        ident_comm_end:

        pop     R17
        pop     R16

ret
```

```
;COMPARE CHAR BY CHAR a command in RAM with a command in program memory
compare_char_by_char:

        push    R16
        push    R17
        push    R18
        push    R19

        ldi     R19,    0
        ldi     R18,    3
        push    XH
        push    XL

        char_by_char_loop:
                ld      R16,    X+
                lpm     R17,    Z+

                cp      R16,    R17
                brne    char_by_char_KO

                dec     R18
                cpi     R18,    0
                breq    char_by_char_OK
                rjmp    char_by_char_loop

        char_by_char_OK:
        ori     flags,  0b00000010
        rjmp    char_by_char_end

        char_by_char_KO:
        add     ZL,     R18
        adc     ZH,     R19
        rjmp    char_by_char_end

        char_by_char_end:

        pop     XL
        pop     XH

        pop     R19
        pop     R18
        pop     R17
        pop     R16

ret


;*********************************************************************
;*** MULTIPLICACIÓN DE DOS NÚMEROS en RAM
;*********************************************************************
;INPUTs:
;                       the numbers are taken from the RAM
```

```
;OUTPUTs:
;                       the result is stored in RAM
;*********************************************************************

mult_uriol:

        push    XH
        push    XL
        push    YH
        push    YL
        push    ZH
        push    ZL


        ldi     XH,     high(num1)
        ldi     XL,     low(num1)
        ldi     YH,     high(num2)
        ldi     YL,     low(num2)
        ldi     ZH,     high(result)
        ldi     ZL,     low(result)

        ldi     R17,    2*NumBytes
        ldi     R16,    0
        result_zero:
                st      Z+,     R16
                dec     R17
                breq    result_zero_end
                rjmp    result_zero
        result_zero_end:

        ldi     ZH,             high(int_mul)
        ldi     ZL,             low(int_mul)
        ldi     mcarry,         0       ; carry=0
        ldi     front_zeros,    NumBytes
        ldi     back_zeros,     0

 multiply:
                ldi     XH,             high(num1)
                ldi     XL,             low(num1)
                ldi     front_zeros,    NumBytes+1
                mov     R17,            back_zeros

                ldi     R16,            0
                fill_back_zeros:
                        cpi     back_zeros,     NumBytes
                        breq    add_int_muls
                        cpi     R17,            0
                        breq    fill_back_zeros_end
                        st      Z+,             R16
                        dec     R17
                        rjmp    fill_back_zeros
          fill_back_zeros_end:
```

```
        ld      R17,            Y+
        inc     back_zeros
        ldi     mcarry,         0

        fill_int_mul:

                cpi     front_zeros,    1
                brne    fill_int_mul_1
                ldi     R16,            0
                rjmp    fill_last_one

                fill_int_mul_1:
                        ld      R16,    X+
                fill_last_one:
                        mul     R16,    R17
                        mov     R18,    R0
                        add     R18,    mcarry
                        mov     mcarry, R1
                        brcc  skip_2
                        inc     mcarry

                skip_2:
                        st      Z+,     R18
                        dec     front_zeros
                        breq    fill_int_mul_end
        rjmp    fill_int_mul
        fill_int_mul_end:

                ldi     R16,    NumBytes
                mov     R17,    back_zeros
                clc
                sbc     R16,    R17

        ldi     R17,    0
        fill_front_zeros:
                cpi     R16,    0
                breq    fill_front_zeros_end
                st      Z+,     R17
                dec     r16
                rjmp    fill_front_zeros
                fill_front_zeros_end:

        rjmp    multiply


add_int_muls:

        ldi     XH,     high(int_mul)
        ldi     XL,     low(int_mul)
        CLC
        LDI     R19,    NumBytes+1
```

```
                RELOAD:
                        LDI     R18,    2*NumBytes
                        ldi     ZH,     high(result)
                        ldi     ZL,     low(result)
                        DEC     R19
                        BREQ    mult_uriol_end

                KEEP_ADDING:
                        LD      R16,    X+
                        LD      R17,    Z
                        ADC     R16,    R17
                        ST      Z+,     R16
                        DEC     R18
                        BREQ    RELOAD
                RJMP    KEEP_ADDING

        mult_uriol_end:

        pop     ZL
        pop     ZH
        pop     YL
        pop     YH
        pop     XL
        pop     XH

ret


;INPUTS:
;                               R20     :       size of the number expressed bytes
;                               R21     :       number of shifts to make
;                               XH:XL   :       the RAM direction of the number
;OUTPUTS:
;                               The number in RAM, now shifted
shift_RAM_right:

        push    R16
        push    R17
        push    R18
        push    R19
        in      R16,    SREG
        push    R16
        push    YH
        push    YL

        mov     YH,     XH
        mov     YL,     XL
        mov     R19,    R20

        shift_RAM_right_loop_2:
        ldi     R18,    0
```

```
mov     XH,     YH
mov     XL,     YL
mov     R20,    R19

        shift_RAM_right_loop_1:
                ld      R16,    X+

                cpi     R20,    1
                breq    shift_RAM_right_skip1

                ;save the LSB
                ld      R17,    X
                andi    R17,    0b00000001
                lsl     R17
                lsl     R17
                lsl     R17
                lsl     R17
                lsl     R17
                lsl     R17
                lsl     R17
                rjmp    shift_RAM_right_skip2

                shift_RAM_right_skip1:
                ldi     R17,    0

                shift_RAM_right_skip2:
                ;shift
                lsr     R16

                ;add the last LSB saved
                add     R16,    R17

                ;save the new RAM value
                sbiw    XH:XL,  1
                st      X+,     R16

                ;now there's a new LSB saved
                ;mov    R18,    R17

                ;are we there yet?
                dec     R20
                breq    shift_RAM_right_loop_1_end
                rjmp    shift_RAM_right_loop_1

        shift_RAM_right_loop_1_end:
        dec     R21
        breq    shift_RAM_right_loop_2_end
        rjmp    shift_RAM_right_loop_2

shift_RAM_right_loop_2_end:

pop     YL
```

```
        pop     YH
        pop     R16
        out     SREG,   R16
        pop     R19
        pop     R18
        pop     R17
        pop     R16

ret



;VALIDAR SI UN CARACTER ASCII REPRESENTA UN NÚMERO ENTRE 0 Y 9
;INPUTs:        R16(caracter ASCII de 8 bits)
;OUTPUTs        :       flags(number)
validar_num_ascii:

                cpi             R16,            '0'
                brlo            invalido
                cpi             R16,            '9'+1
                brsh            invalido

                ori             flags,          0b00001000
                rjmp            valido

                invalido:
                andi            flags,          0b11110111

                valido:

ret


config_ADC:

        ldi     R16,            0b00000000
        sts     ADCSRB,         R16

        ldi     R16,            0b01000000
        sts     ADMUX,          R16

        ldi     R16,            0b00111110
        sts     DIDR0,          R16

        ;ADC APAGADO AL INICIO
        ldi     R16,            0b01101011
        sts     ADCSRA,         R16

ret
```

```
;CONFIGURAR USART:
;INPUTs:
;OUTPUTs        :       UCSRA,UCSRB,UCSRC,UBRR
CONFIG_USART:

        push    R16

                LDI     R16             ,high(Fosc/bps/16-1)    ;$00
                sts     UBRR0H          ,R16
                LDI     R16             ,low(Fosc/bps/16-1)     ;$0C
                sts     UBRR0L,R16
                LDI     R16             ,0B00000000
                sts     UCSR0A          ,R16
                ldi     R16             ,0B10011000
                sts     UCSR0B          ,R16
                ldi     R16             ,0B00101110
                sts     UCSR0C          ,R16
                pop     R16

RET

;RECIBIR DATO:
;INPUTs:
;OUTPUTs        :       R20,UDR0
RX:
    push        r16

        RX_loop:
                lds     R16,    UCSR0A
                andi    R16,    0b10000000
                cpi     R16,    0b10000000
                breq    RX_read
                RJMP    RX_loop
        RX_read:
                lds     DATO,   UDR0

                pop     R16

RET


;TRANSMITIR DATO:
;INPUTs         :       DATO
;OUTPUTs        :       UDR0
TX:
                push    R16

  TX_loop:
                lds     R16,    UCSR0A
                andi    R16,    0b00100000
                cpi     R16,    0b00100000
                breq    TX_write
```

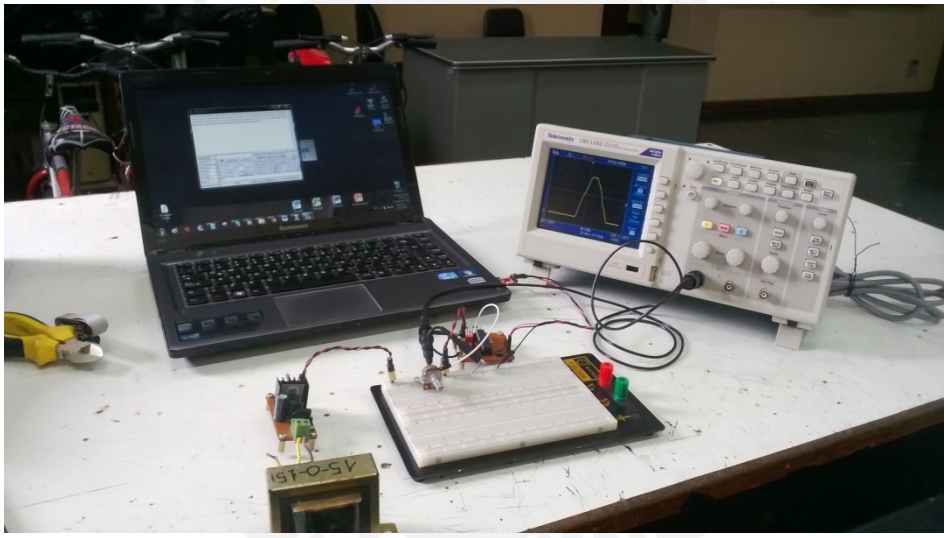
```
                RJMP    TX_loop
        TX_write:
                sts     UDR0,   DATO
                pop     R16

RET



;ENVÍA UN CARRIAGE RETURN Y UN LINE FEED POR EL USART
;INPUTs:
;OUTPUTs        :
enter:

                push            DATO

                ldi             DATO,   $0D
                rcall           TX
                ldi             DATO,   $0A
                rcall           TX

                pop             DATO

ret

;ENVIAR CADENA DE CARACTERES:
;INPUTs         :       DATO
;OUTPUTs        :       UDR,R16,Z
ENVIACADENA:
                LPM     DATO,   Z+
                CPI     DATO,   0
                BREQ    END
                RCALL   TX
                RJMP    ENVIACADENA
        END:

RET


ADR:    .db     "ADR"
TAR:    .db     "TAR"
TAS:    .db     "TAS"
AVG:    .db     "AVG"
```

ANEXO C

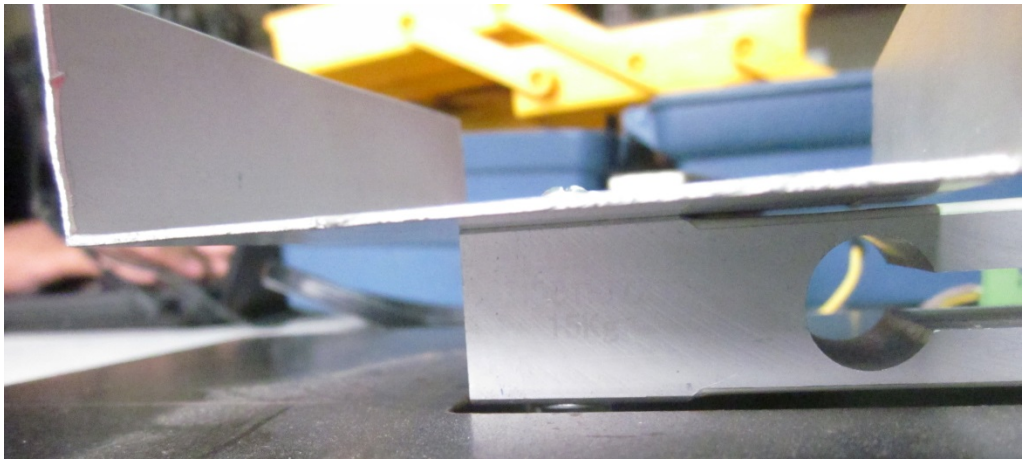
Fotos de equipos usados.

BALANZA DE REFERENCIA PARA MEDIR PESOS DE PRUEBA.

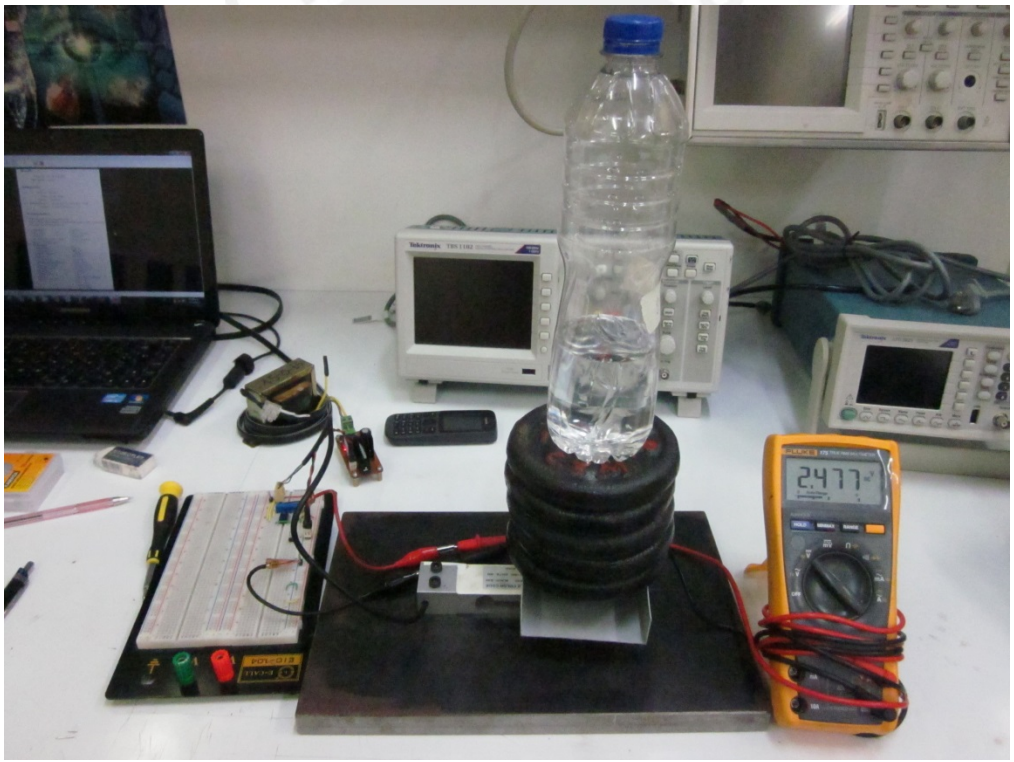
PRUEBAS DE LA TERCERA Y CUARTA ETAPA USANDO UN POTENCIÓMETRO PARA GENERAR LA SEÑAL DE ENTRADA AL ADC.



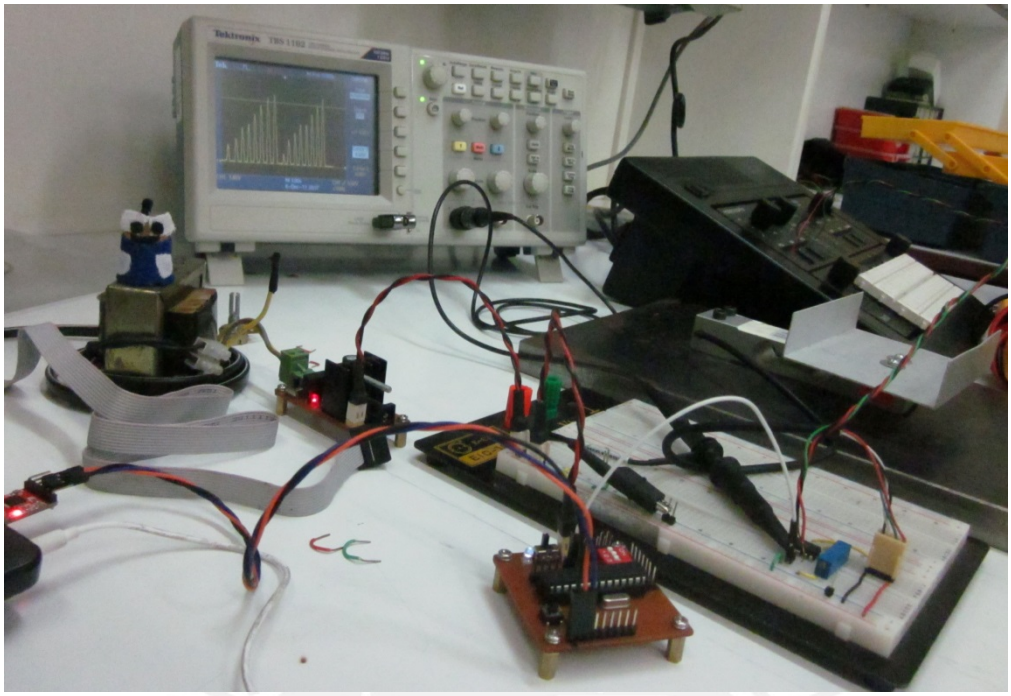
UNIÓN DE LA CELDA DE CARGA Y SU PLATAFORMA DE PRUEBAS



PRUEBAS DE LAS CUATRO PRIMERAS ETAPAS USANDO PESOS DE PRUEBA



PRUEBAS DE LAS CUATRO PRIMERAS ETAPAS PRESIONANDO LA PLATAFORMA DE PRUEBA



GENERADOR DE SEÑALES USADO PARA PROBAR LA UNIFORMIDAD DEL RESULTADO DEL SISTEMA

ANEXO D

Hojas de datos de los componentes principales en el diseño y las pruebas. [ver CD adjunto]

- ATmega88PA
- MAX491
- Celda de carga: XS-L6D
- Celda de carga: 1130
- INA126
- INA826