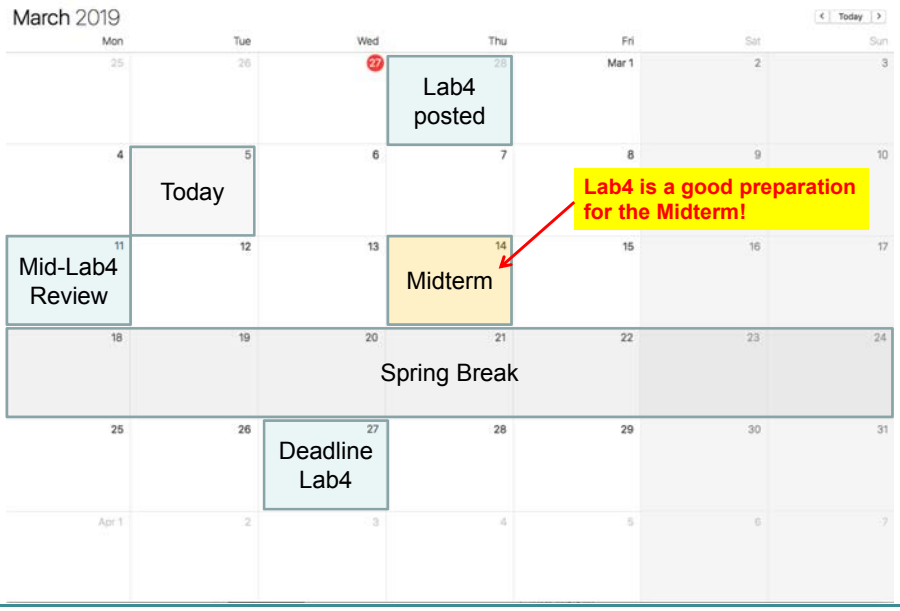
Embedded Systems

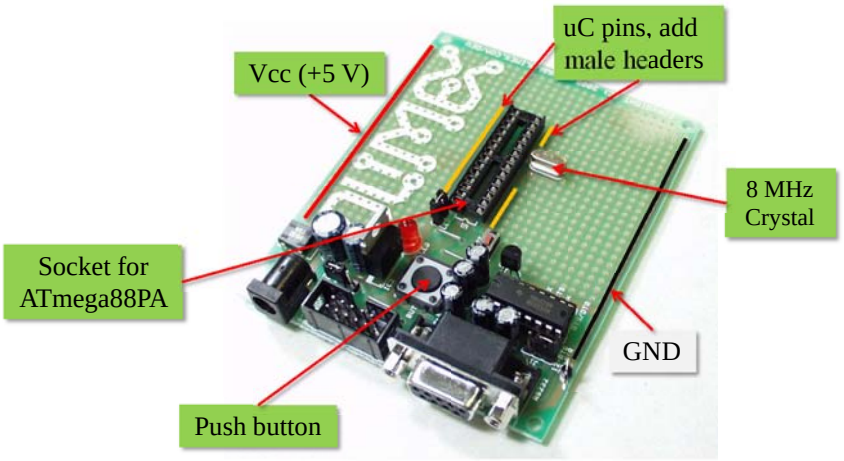
Lab 4 Considerations

Timeline – Lab4 and Midterm



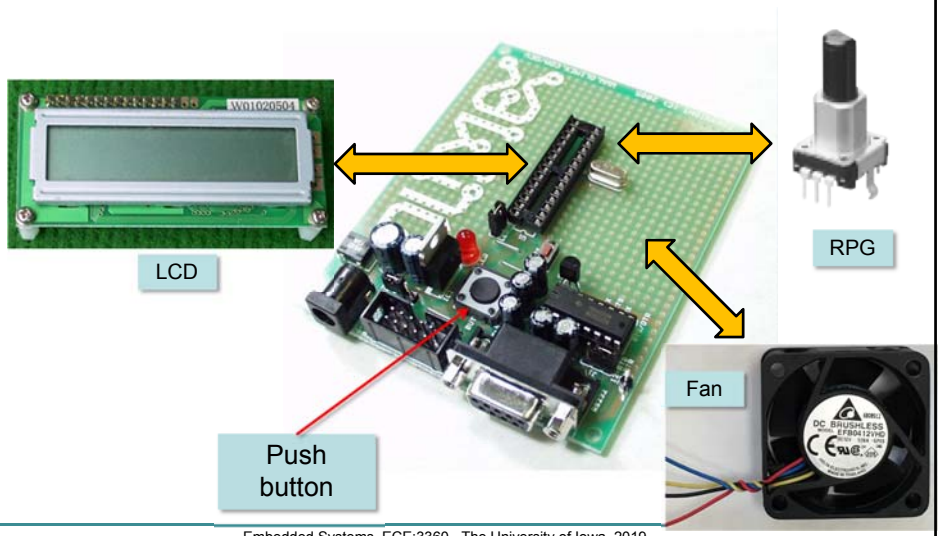
28-Pin AVR Development Board

Starting with Lab 4, we will start using a "bigger" controller → get "**Kit B**" from the Electronic Shop



Lab 4 Overview

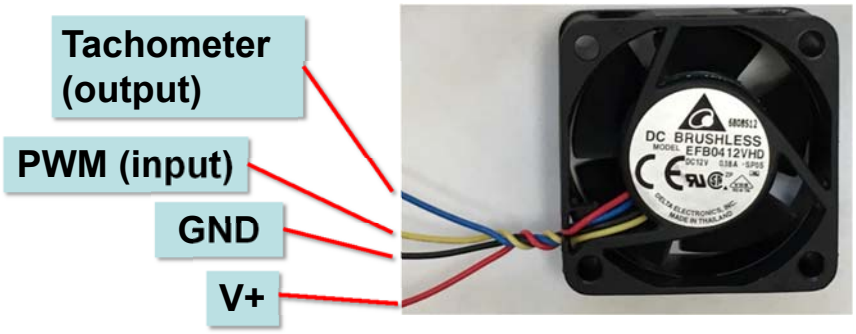
Extend Lab 3 (PWM Generator) by adding a push button, LCD, and fan. The push button allows one to select the different fan monitoring modes.



Fan (EFB0412VHD-SP05)

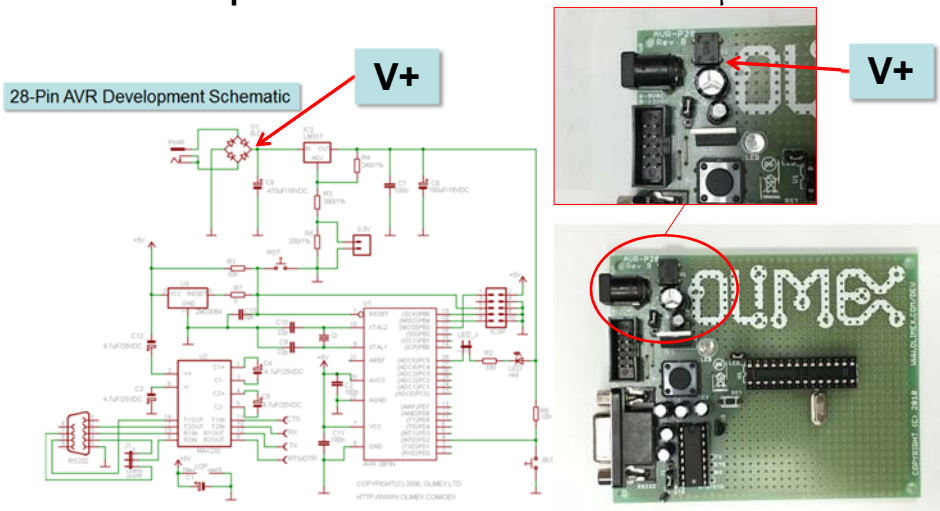
From data sheet (see Resources on website)

- Size / dimension — Square - 40mm L x 40mm H
- Features — PWM control; tachometer
- Power (Watts) — 1.44W
- RPM — 9000 RPM
- Current rating — 0.120A
- Voltage range — 5 ~ 13.8VDC → we will use 7.8 V (V+) → 5600 rpm



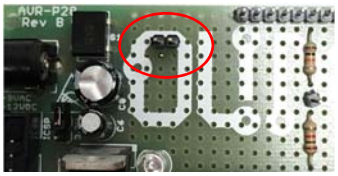
Supply Voltage for Fan

- We will use an unregulated voltage (V+) of ~ 7.8V for operating the fan (9 V wall wart power supply)
- This will require a small modification of the μC board

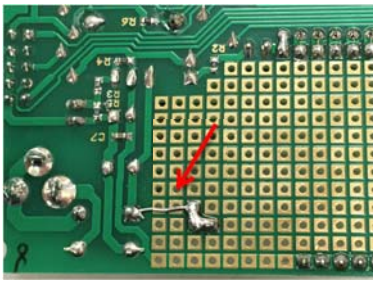


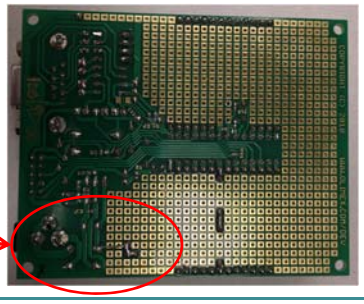
Supply Voltage for Fan

1) Install a single row male header



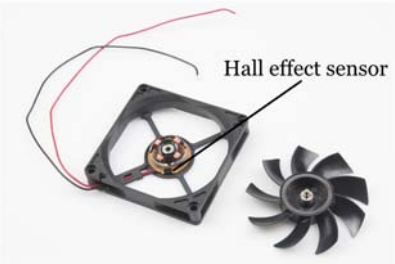
2) Install a bridge between header and + pin of rectifier

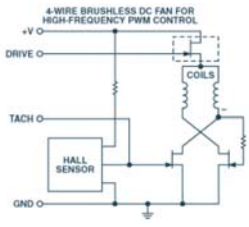




Fan - Tachometer Signal

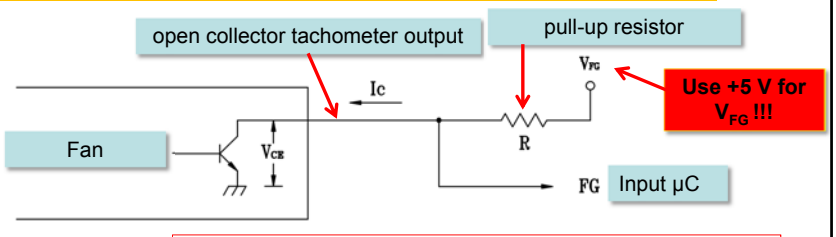
- **A Hall sensor is used to generate a signal proportional to the speed of the fan**
- **Can be several pulses per revolution**
- **Examples:**



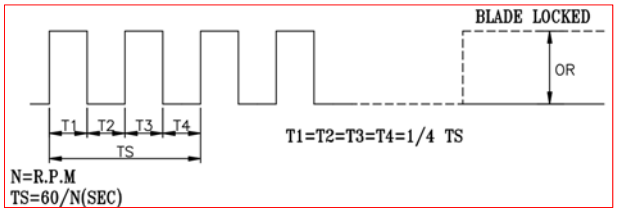


Tachometer Signal

Interfacing the tachometer output to the µC requires a pull-up resistor



Output signal: 2 pulses per revolution



Design of R → From the data sheet:

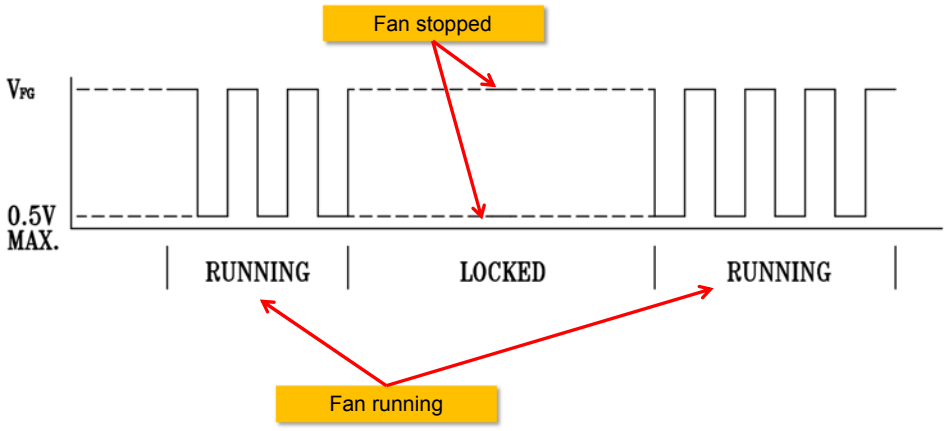
V_{CE} (sat)=0.5V MAX. V_{FG} =13.8VDC MAX.

I_C =5mA MAX. $R \geq V_{FG}/I_C$

Use +5 V for V_{FG} !!!

Tachometer Signal

- **Note: when the fan stops (locked) the tachometer signal (V_{FG}) can be logic 1 or 0**



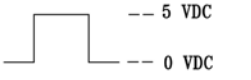
Fan Speed & PWM

- The speed of the fan can be changed by using PWM
- The Table below gives an example for the relation between duty cycle of PWM signal and speed
- Note: this is supply voltage dependent!

SPEED VS PWM CONTROL SIGNAL:
(AT 25°C, RATED VOLTAGE & PWM SIGNAL AS FOLLOW)

DUTY CYCLE (%)	SPEED R.P.M.	CURRENT (A) TYP.
100	9000±10%	0.12
55	5000±10%	0.05
25	2000±250	0.03
0	0	0.02

* PWM SIGNAL
PWM FREQUENCY = 25KHz

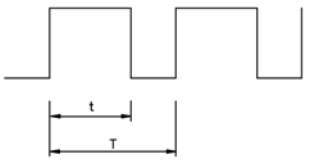


- MIN. START DUTY CYCLE : 25%.
  WHEN DUTY CYCLE IS SET FOR MORE THAN 25%, THE FAN WILL BE ABLE TO START FROM A DEAD STOP.

Fan Speed & PWM

PWM signal specification → From the data sheet:

PWM CONTROL SIGNAL:
SIGNAL VOLTAGE RANGE: 0 ~ 20 VDC



------ HIGH LEVEL : 20 VDC MAX. 2.8 VDC MIN.

------ LOW LEVEL : 0.8 VDC MAX. 0 VDC MIN.

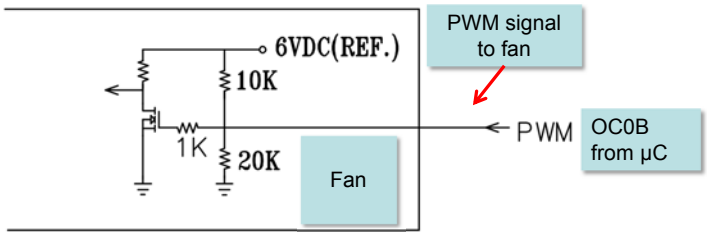
DUTY CYCLE= $\frac{t}{T}$ *100(%)

- THE FREQUENCY FOR CONTROL SIGNAL OF THE FAN SHALL BE ABLE TO ACCEPT 30HZ~300KHZ.
- FOR REDUCING THE SWITCHING NOISE, THE PREFERRED OPERATING POINT FOR THE FAN IS 25KHZ.
- AT 100% DUTY CYCLE,THE ROTOR WILL SPIN AT MAXIMUM SPEED.
- AT 0% DUTY CYCLE,THE ROTOR WILL STOP SPIN .
- WITH CONTROL SIGNAL LEAD DISCONNECTED,THE FAN WILL SPIN AT MAXIMUN SPEED.

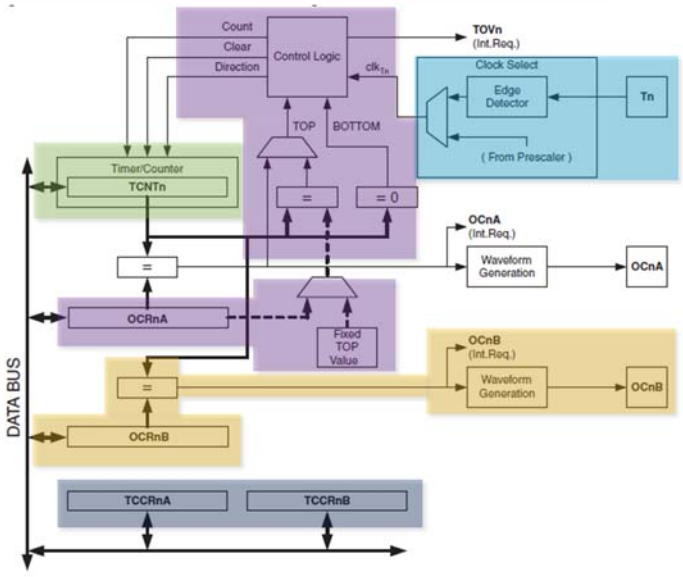
We will use a PWM frequency of 40 kHz → TC configuration

PWM Signal

- Use Timer/Counter0 (TC0 → 8 bit) of the ATmeag88 to generate the PWM signal for the fan (alternative: 8-bit TC2)
- Must be done by utilizing Waveform Generation functionality of the TC0 (→ pin OC0B)
- Once configured correctly, the PWM is generated fully automated
- Need to set compare registers OCR0A and OCR0B to set frequency and duty cycle (see Atmega88 data sheet)



8-Bit Counter/Timer – Basic Components



TC0 - PWM Function Generation

- The fast Pulse Width Modulation or fast PWM mode (WGM02:0 = 3 or 7) provides a high frequency PWM waveform generation option.
- In fast PWM, the counter counts from BOTTOM to TOP then restarts from BOTTOM.
- TOP is defined as 0xFF when WGM02:0 = 3, and OCR0A when WGM02:0 = 7.

Table 15-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on (1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

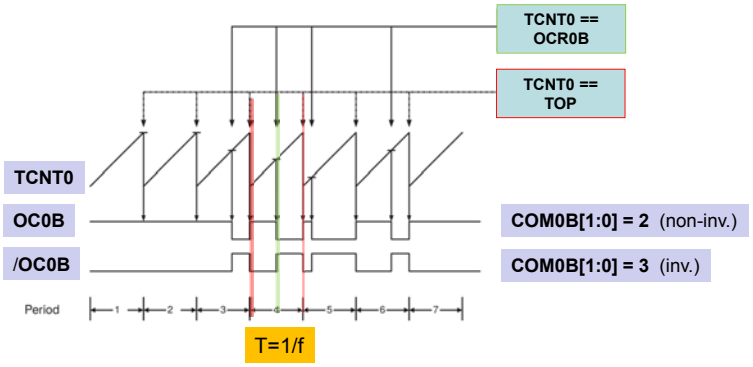
Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

TC0 - PWM Function Generation

- Fast PWM is well suited for power regulation, rectification, and DAC applications.
- High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.
- For a description of other PWM functionality → see ATmega88 data sheet
  - E.g., PWM, phase correct

TC0 - PWM Function Generation

- In non-inverting Compare Output mode, the Output Compare (OC0B) is cleared on the compare match between TCNT0 and OCR0B, and set at BOTTOM.
- In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM.



TC0 - PWM Compare Output Modes

- In non-inverting Compare Output mode, the Output Compare (OC0B) is cleared on the compare match between TCNT0 and OCR0B, and set at BOTTOM.
- In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM.

Table 15-6. Compare Output Mode, Fast PWM Mode(1)

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on Compare Match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on Compare Match, clear OC0B at BOTTOM, (inverting mode).

Note: I/O pin must be configured as output!

TC0 – Clock selection

- **Don't forget to set the clock source for TC0!**

Table 15-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

TC0 - Configuration

- **TC0 configuration is done in two control registers**

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TC0 compare output Mode selection

TC0 mode selection

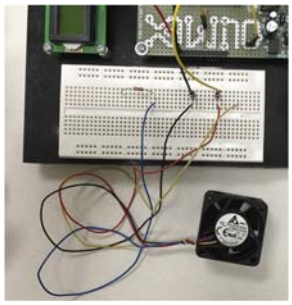
TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TC0 clock selection

Lab 4 - Approach

- HW/SW components of Lab 4:
  - RPG
  - Push button
  - LCD
  - PWM generation for Fan
  - Tachometer signal (speed sensing)
- Use interrupts and µC HW!
- Organize software in subroutines and ISRs
- Draw a wiring diagram
- Se a TA for fan check out



Lab 4 LCD

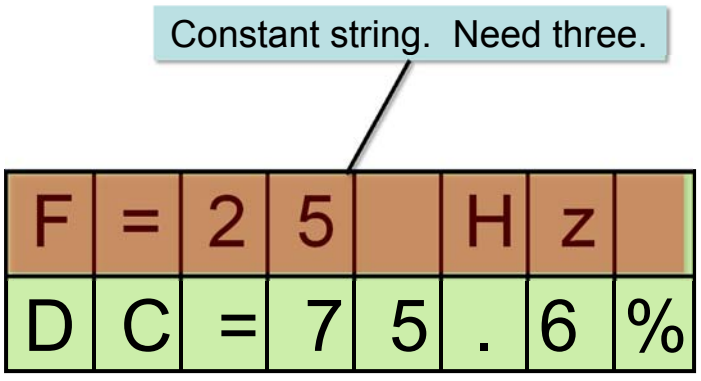
Character Formatting

Formatting Example

- **For displaying information (duty cycle, mode, state, rps, etc.) on the LCD, some formatting is required, similarly to the example given below**

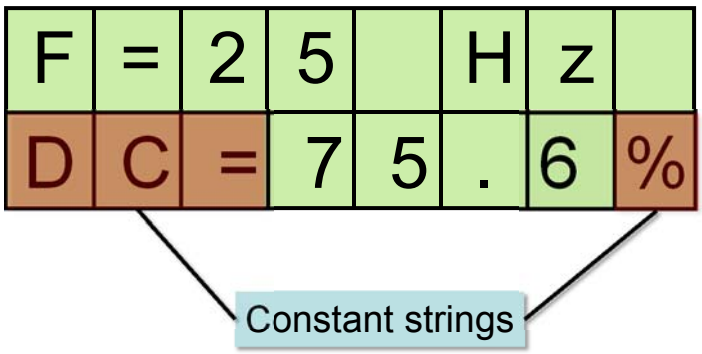
F	=	2	5		H	z	
D	C	=	7	5	.	6	%

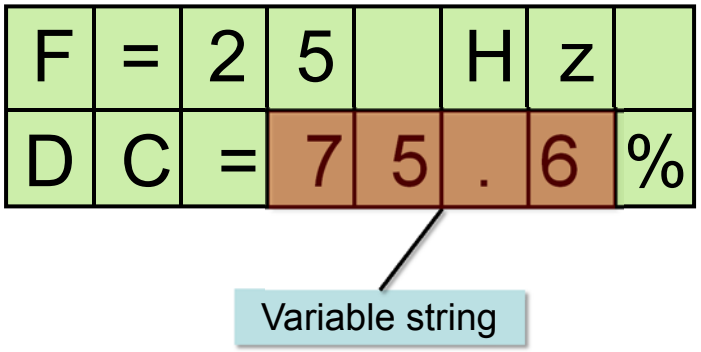
Note: in this example, we assume 3 predefined (selectable) frequencies, but we have a variable duty cycle!

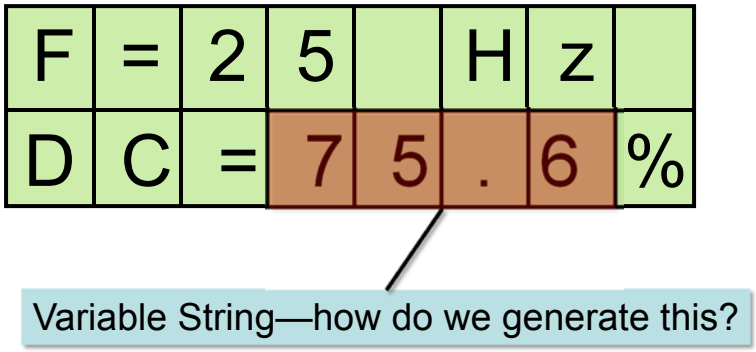


Displaying Constant Strings

```
sf25: .DB "F=25 Hz "   ; Create a static string in program memory.
…
…
    ldi    r24,8              ; r24 <-- length of the string
    ldi    r30,LOW(2*sf25)    ; Load Z register low
    ldi    r31,HIGH(2*sf25)   ; Load Z register high
    rcall displayCString

displayCString:
L20:
    lpm                       ; r0 <-- first byte
    swap   r0                 ; Upper nibble in place
    out    PORTC,r0           ; Send upper nibble out
    rcall LCDStrobe           ; Latch nibble
    rcall _delay_100u         ; Wait
    swap   r0                 ; Lower nibble in place
    out    PORTC,r0           ; Send lower nibble out
    rcall LCDStrobe           ; Latch nibble
    rcall _delay_100u         ; Wait
    adiw   zh:zl,1            ; Increment Z pointer
    dec    r24                ; Repeat until
    brne   L20                ; all characters are out
    ret
```


F = 2 5 H z
D C = 7 5 . 6 %
Variable string


F = 2 5 H z
D C = 7 5 . 6 %
Variable String—how do we generate this?

Formatting for LCD

Problem

Given a 16 bit (two byte) variable

```
DC_HI:DC_LO
```

which contains a (binary) duty cycle value in the range 0 thru 999, generate the proper display string to show this value on the LCD in the form:

```
xy.z
```


ASCII

The **American Standard Code for Information Interchange** (acronym: **ASCII**) is a character-encoding scheme based on the ordering of the English alphabet.

ASCII codes represent text in computers, communications equipment, and other devices that use text.

Most modern character-encoding schemes are based on ASCII, though they support many more characters than do plain ASCII.

The LCD display uses ASCII character codes. To display, for example, the digit '5', one sends '5's ASCII code to the display.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

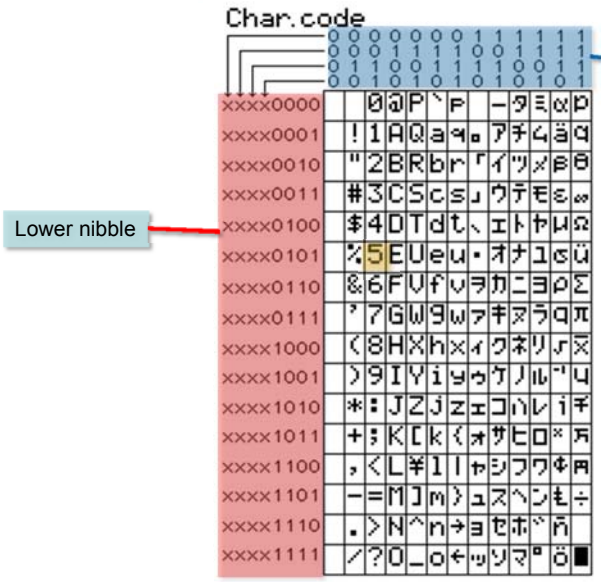
Embedded Systems, ECE:3360. The University of Iowa, 2019
Lab 4 Slide 32

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

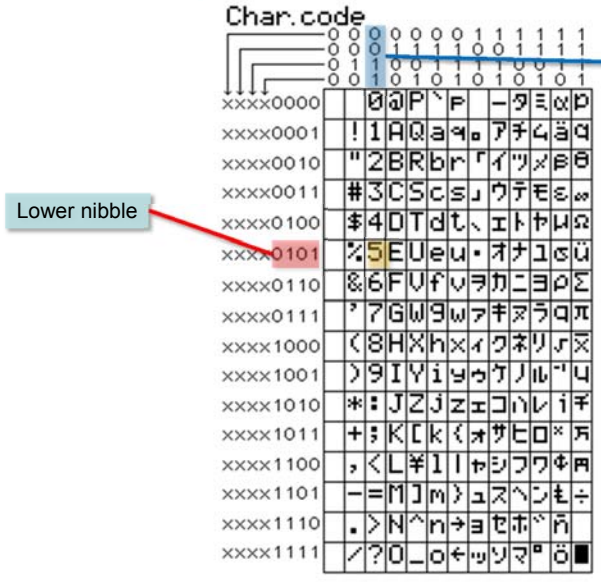
Embedded Systems, ECE:3360. The University of Iowa, 2019
Lab 4 Slide 34

HD44780 Character Codes



Question: what is the character code for '5'

HD44780 Character Codes



Question: what is the character code for '5' ?

Answer: `00110101`

Answer: `0x35`

Some ASCII "Tricks of the Trade"

The decimal digits 0-9 are represented by the ASCII codes 0x30-0x39 respectively.

Dec	Hex	Oct	Html	Char
48	30	060	&#48;	0
49	31	061	&#49;	1
50	32	062	&#50;	2
51	33	063	&#51;	3
52	34	064	&#52;	4
53	35	065	&#53;	5
54	36	066	&#54;	6
55	37	067	&#55;	7
56	38	070	&#56;	8
57	39	071	&#57;	9

So simply add 0x30 to convert the number to its ASCII code: 5 + 0x30 = 0x35

Some ASCII "Tricks of the Trade"

The letters 'A-Z' are represented by the ASCII codes 65-90 respectively, while the lowercase letters a-z are represented by the ASCII codes 97-122 respectively.

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	&#64;	@	96	60	140	&#96;	`
65	41	101	&#65;	A	97	61	141	&#97;	a
66	42	102	&#66;	B	98	62	142	&#98;	b
67	43	103	&#67;	C	99	63	143	&#99;	c
68	44	104	&#68;	D	100	64	144	&#100;	d
69	45	105	&#69;	E	101	65	145	&#101;	e

Thus, to convert from letters from upper- to lower case, add 0x20.

Isolating Digits of the Duty Cycle

Consider : **`DC_HI:DC_LO = 765`**

Step 1. Divide **`DC_HI:DC_LO`** by 10: 765/10 = 76 with a remainder of 5. Thus 5 is the rightmost digit.

Step 2. Determine the ASCII code for 5. The digits 0-9 are represented by the ASCII codes 0x30-0x39 respectively. So simply add 0x30: 5 + 0x30 = 0x35

Step 3. Place a decimal point (0x2e) to the left of the rightmost ASCII character

Step 4. Now divide the quotient from step 1) by 10: 76/10 = 7 with a remainder of 6. So the next digit (to the left of the decimal point) is 6 and the corresponding ASCII code is 6 + 0x30 = 0x36

Step 5. The quotient from Step 4 is the leftmost digit of the duty cycle string: 7 + 0x30 = 0 x37

So the display String is: 0x37, 0x36, 0x2e, 0x35

Isolating Digits of the Duty Cycle

- **How do we do the division by 10?**
- **AVR instruction set does not include a divide instruction**
- **There is no obvious way to isolate the digits of the duty cycle without doing a division operation**
- **A lookup table would be very "expensive"**
  - **Would require 1000 entries.**

Isolating Digits of the Duty Cycle

You can make use of a library of prewritten math subroutines. These are on the Atmel website. See Application Note “AVR200: Multiply and Divide Routines”.

The following sample code uses the routine **div16u** for unsigned, 16-bit division. It uses these registers

```
.def    drem16uL = r14
.def    drem16uH = r15
.def    dres16uL = r16
.def    dres16uH = r17
.def    dd16uL   = r16
.def    dd16uH   = r17
.def    dv16uL   = r18
.def    dv16uH   = r19
```

Usage is very simple. Load the number you want to divide (say 567) into dd16uH:dd16uL and the number you want to divide by (say 10) into dv16h:dv16L and then call div16u. The result is placed in "dres16uH:dres16uL" and the remainder in "drem16uH:drem16uL".

```
; Divide (unsigned) 567 by 10
   ldi dd16uL,low(567)  ; Dividend
   ldi dd16uH,high(567) ; Dividend
   ldi dv16uL,low(10)   ; Divisor
   ldi dv16uH,high(10)  ; Divisor
   rcall div16u         ; Result (56)in dres16uH:dres16uL
                        ; Remainder (7) in drem16uH:drem16uL
```



```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start
…
; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00
start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

Driver for sample code LCD write code

Create static strings in flash memory, but make sure we don’t start executing the strings…

Note how we terminate the strings with nulls – this signals the end of the string for the display routine.

Also, the number of bytes must be even (why?) otherwise the assembler will complain and pad the string.

Code snippets!!!

```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start
…
; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00

start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

Driver for sample code LCD write code

Initialize the LCD properly


```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start
…
; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00

start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

Driver for sample code LCD write code

The **displayCString** routine, described later, expects the **Z**-pointer register to point to the start of the string. This is the standard AVR idiom for loading **Z** with a string in flash

```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start
…
; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00

start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

Driver for sample code LCD write code

Call **displayCString** routine, which will send the string msg1 out, up to the terminating null byte.

The LCD shows:


```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start
…
; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00

start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

Driver for sample code LCD write code

The user changes the duty cycle, so this part of the LCD code must be created dynamically.

Assume the user has selected a duty cycle of 56.9%.

```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start

; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00
…
start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

Driver for sample code LCD write code

The **displayDC** routine expects the duty cycle in the **R26:R25** (MSB:LSB) register pair.

In this example, we hardcode the 569 which represents 56.9%

In Lab 4 code the 569 will be a variable, perhaps a register pair.


```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start
…
; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00

start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

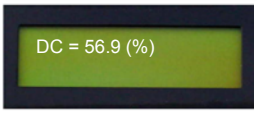
Driver for sample code LCD write code

Display the duty cycle. The LCD shows:

```
.include "m88padef.inc"
.cseg
.org 0x00     ; PC points here after reset
    rjmp start
…
; Create static strings in program memory.
   msg1: .db "DC = ",0x00
   msg2: .db " (%) ",0x00

start:
; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Display the string: "DC = "
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high
   rcall displayCString

; Display the duty cycle.
   ldi   r25,low(569)
   ldi   r26,high(569)
   rcall displayDC

; Display the string " (%)"
   ldi   r30,LOW(2*msg2)    ; Load Z register low
   ldi   r31,HIGH(2*msg2)   ; Load Z register high
   rcall displayCString
```

Driver for sample code LCD write code

Display the second static string: (%). The LCD shows:




```
displayDC:

.dseg
   dtxt:  .BYTE 5       ; Allocation

.cseg
   mov   dd16uL,r25     ; LSB of number to display
   mov   dd16uH,r26     ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00       ; Terminating NULL
   sts   dtxt+4,r20     ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u     ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20        ; Convert to ASCII
   sts   dtxt+3,r14     ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e       ; ASCII code for .
   sts   dtxt+2,r20     ; Store in RAM
…
```

Code to create a string in data memory (RAM).

This code formats a 3-digit number as a floating point number. For example

569 → 56.9

It peels of the digits by successively dividing by 10 and storing results in reverse order (why?).

Another routine will then send out these bytes to the LCD.

```
displayDC:

.dseg
   dtxt:  .BYTE 5        ; Allocation

.cseg
   mov   dd16uL,r25      ; LSB of number to display
   mov   dd16uH,r26      ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00        ; Terminating NULL
   sts   dtxt+4,r20      ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u      ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20         ; Convert to ASCII
   sts   dtxt+3,r14      ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e        ; ASCII code for .
   sts   dtxt+2,r20      ; Store in RAM
…
```

Create 5 bytes in RAM to hold the string. Note the use of the **.dseg** assembler directive.


```
displayDC:

.dseg
   dtxt:  .BYTE 5        ; Allocation

.cseg
   mov   dd16uL,r25      ; LSB of number to display
   mov   dd16uH,r26      ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00        ; Terminating NULL
   sts   dtxt+4,r20      ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u      ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20         ; Convert to ASCII
   sts   dtxt+3,r14      ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e        ; ASCII code for .
   sts   dtxt+2,r20      ; Store in RAM
…
```

Note the use of the **.cseg** assembler directive. The following lines are code and not data.

```
displayDC:

.dseg
   dtxt:  .BYTE 5       ; Allocation

.cseg
   mov   dd16uL,r25     ; LSB of number to display
   mov   dd16uH,r26     ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00       ; Terminating NULL
   sts   dtxt+4,r20     ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u     ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20        ; Convert to ASCII
   sts   dtxt+3,r14     ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e       ; ASCII code for .
   sts   dtxt+2,r20     ; Store in RAM
…
```

Copy the number we want to display from **R25:R26** (that the caller set up) to the registers the 16-bit unsigned divide routine uses


```
displayDC:

.dseg
   dtxt:  .BYTE 5       ; Allocation

.cseg
   mov   dd16uL,r25     ; LSB of number to display
   mov   dd16uH,r26     ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00       ; Terminating NULL
   sts   dtxt+4,r20     ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u     ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20        ; Convert to ASCII
   sts   dtxt+3,r14     ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e       ; ASCII code for .
   sts   dtxt+2,r20     ; Store in RAM
…
```

The algorithm successively divide by 10 – set this up

```
displayDC:

.dseg
   dtxt:  .BYTE 5        ; Allocation

.cseg
   mov   dd16uL,r25      ; LSB of number to display
   mov   dd16uH,r26      ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00        ; Terminating NULL
   sts   dtxt+4,r20      ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u      ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20         ; Convert to ASCII
   sts   dtxt+3,r14      ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e        ; ASCII code for .
   sts   dtxt+2,r20      ; Store in RAM
…
```

Store the terminating null.

Notice how we use the assembler to calculate the address for us when it assembles the code


```
displayDC:

.dseg
   dtxt:  .BYTE 5        ; Allocation

.cseg
   mov   dd16uL,r25      ; LSB of number to display
   mov   dd16uH,r26      ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00        ; Terminating NULL
   sts   dtxt+4,r20      ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u      ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20         ; Convert to ASCII
   sts   dtxt+3,r14      ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e        ; ASCII code for .
   sts   dtxt+2,r20      ; Store in RAM
…
```

Perform the 16-bit division by 10

```
displayDC:

.dseg
   dtxt:  .BYTE 5       ; Allocation

.cseg
   mov   dd16uL,r25     ; LSB of number to display
   mov   dd16uH,r26     ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00       ; Terminating NULL
   sts   dtxt+4,r20     ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u     ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20        ; Convert to ASCII
   sts   dtxt+3,r14     ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e       ; ASCII code for .
   sts   dtxt+2,r20     ; Store in RAM
…
```

Add 0x30 to the remainder to convert to the digits ASCII code


```
displayDC:

.dseg
   dtxt:  .BYTE 5       ; Allocation

.cseg
   mov   dd16uL,r25     ; LSB of number to display
   mov   dd16uH,r26     ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00       ; Terminating NULL
   sts   dtxt+4,r20     ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u     ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20        ; Convert to ASCII
   sts   dtxt+3,r14     ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e       ; ASCII code for .
   sts   dtxt+2,r20     ; Store in RAM
…
```

Store it in RAM. Note that we put this at the back of the string.

```
displayDC:

.dseg
   dtxt:  .BYTE 5        ; Allocation

.cseg
   mov   dd16uL,r25      ; LSB of number to display
   mov   dd16uH,r26      ; MSB of number to display

   ldi   dv16uL,low(10)
   ldi   dv16uH,high(10)

; Store terminating for the string.
   ldi   r20,0x00        ; Terminating NULL
   sts   dtxt+4,r20      ; Store in RAM

; Divide the number by 10 and format remainder.
   rcall div16u      ; Result: r17:r16, rem: r15:r14
   ldi   r20,0x30
   add   r14,r20         ; Convert to ASCII
   sts   dtxt+3,r14      ; Store in RAM

; Generate decimal point.
   ldi   r20,0x2e        ; ASCII code for .
   sts   dtxt+2,r20      ; Store in RAM
…
```

Place decimal point

Continue until the whole string is in place


```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Displays a dynamic character string, stored
;; in data (RAM), on the LCD.  The string
;; is assumed to be null-terminated.
;;
;; Call:
;;
;;   ldi   r30,LOW(dtxt)   ; Load Z register low
;;   ldi   r31,HIGH(dtxt)  ; Load Z register high
;;   rcall displayDstring
;;
;;
displayDstring:
   ld    r0,Z+
   tst   r0              ; Reached end of message ?
   breq  done_dsd        ; Yes => quit
   swap  r0              ; Upper nibble in place
   out   PORTC,r0        ; Send upper nibble out
   rcall LCDStrobe       ; Latch nibble
   swap  r0              ; Lower nibble in place
   out   PORTC,r0        ; Send lower nibble out
   rcall LCDStrobe       ; Latch nibble
   rjmp  displayDString
done_dsd:
   ret
```

Code to write a string in data memory (RAM) to the LCD.

... EOL