(Library2)

Assembler Library - II

Reference Manual

Table of Contents

Library II Description.

Library II is a set of Routines wrote in Assembler Language for Atmel AVR Family Microcontrollers to be used with Atmel AVRStudio 4.0 developed by **Author Techonology Corp.** These Routines satisfy a minimum requirement to use an Assembler Language of ease way and reliable form, increasing productivity and offering capabilities to work in group of developers and some manner getting more performance in program code size(less bytes used) and more speed competing direct with more high level Languages like a C.

Library II is composited with CONSTANTS,MACROS,ROUTINES and generic no dependent Hardware routines like MATH,PROTOCOLS and dependent Hardware routines like DISPLAY DRIVERS,EEPROM ACCESS,DATA FLASH,ADCs,DACs.

Author Library II routines use a below schematic of flux of program components. The order of steps must be followed strictly to avoid program unpredictable results.

Program steps.

Step	Command	Description
1	.INCLUDE "DEFS\GLOBAL_DEFINITIONS\GLOBDEFS.INC"	Must be a first include and always be included
2	.EQU _SRAM_BOOT_TYPE=_SRAM_NOT_CLEAR .EQU _AVR_CLOCK = 16000000	Compiler and Software settings
3	.INCLUDE "DEFS\M64_FILES\M64DEF.INC" .INCLUDE "DEFS\M64FILES\M64HDC.INC"	Processor definitions and handle interrupts
4	;+----------------------------------- ;| GRAPH ROUTINES ;+----------------------------------- .INCLUDE "GRAPH\LINE\LINE.HUG" .INCLUDE "GRAPH\RASTER\BASE\RASTER_BASE.INC" ;+----------------------------------- ;| EEPROM ROUTINES ;+----------------------------------- .INCLUDE "EEPROMS\AVRE2P\A256_EX.HUG" ;+----------------------------------- ;| PROTOCOL ROUTINES ;+----------------------------------- .INCLUDE "PROTOCOL\AUTHOR\DSFRAME\DSFRAME.HUG"	Generic Library II routines, left only a example.
5	.INCLUDE	All Hardware dependent

	"HARDWARE_DEFINITIONS.INC"	routines are placed inside of this file
6	**.INCLUDE "LOCAL_DEFINITIONS.INC"**	All not Hardware dependent routines are placed inside of this file such as global program Functions, program Modules
7	**_MAIN_INIT:** **call _CPULED_INIT** **call _MOTION_RELAY_INIT**	_MAIN_INIT label must always include and is the first entry point accessed after boot initialization, and here is placed all drives, routines, objects that must be initialized before is used.
8	**_MAIN:**	Here, after _MAIN: label are placed the Main Program that call all routines placed in Hardware_definition.inc or Local_definition.inc, but if you want to use a OOP techniques Hardware dependent routines must be avoid

Prefixes/Suffixes.

Author Libray II routines use below prefixes and suffixes to facility program scope, program reliability and program clarity.

Prefixes/Suffixes	Description
_ **(Underscore)**	Prefix to signalize that a routine or label is inside a Library Ex: Call _DISP_DATA_WRITE
fn_	Prefix placed at all functions in Local_Definitions.inc file meaning a Function
MODULE	Prefix placed at all programs Module routines that use functions defined in Local_Definitions.inc, if an OOP techniques is used all hardware dependent routines must be avoid.
.INC	Suffix placed in file extension meaning generic include file or Library for AVR with less than 8Kbytes of Flash memory
.HUG	Suffix placed in file extension meaning generic include file or Library for AVR with more than 8Kbytes of Flash memory

Registers.

Author Libray II routines registers use the below definitions.

Register	Description
R0..R15	Usually used as scrap registers otherwise specified.

Acc ,R24
AccH,R25
	Acc is used like an Accumulator of 8bits size and AccH is an Accumulator High with 8 bits size, both may be used separated or together forming a 16bit register in macros prefixed by AccAW, this register is preferred way to pass and return values from routines
	Ex:
	  Ldi Acc,23 ; Load Acc with 23
	  Ldi AccH,18 ; Load AccH with 18
	  Ldiaw 1234 ; Load AccH:Acc with 1234

AccT ,R16
AccTH,R17
	AccT is used like a Temporary Accumulator of 8bits size and AccTH is a Temporaty Accumulator High with 8 bits size, both may be used separated or together forming a 16bit register in macros prefixed by AccAWT, this register too is preferred way to pass and return values from routines
	Ex:
	  Ldi AccT,23 ; Load AccT with 23
	  Ldi AccTH,18 ; Load AccTH with 18
	  LdiawT 1234 ; Load AccTH:AccT with 1234

Temp ,R18
TempH,R19
	Temp is used like a Temporary register of 8bits size and TempH is a Temporaty Register High with 8 bits size, both may be used separated or together forming a 16bit register in macros prefixed by TempH, This register may be used to pass or return values from routines but they use must be used with care. They really a temporary scrap register, long time storage values must be avoid.
	Ex:
	  Ldi Temp,23 ; Load Temp with 23
	  Ldi TempH,18 ; Load TempH with 18

Important Files.

Author Libray II important files are further explained below.

File	Description
GLOBDEFS.INC	Is a Global Definitions include file that contain a set of Macros, Contants used for all **Author programs**. And must be always included.
MATHCONS.INC	Is a Global Definitions when a math routine is need, placed on step 4 of program steps.
xxxxDEF.INC	Is a Global Device Definition include file supplied by Atmel AVRStudio 4 generally located at folder \Programs Files\Atmel\AVR Tools\AvrAssembler2\Appnotes Author routines actually support these devices AT90S1200, AT90S2313, AT90S2323, AT90S8515, AT90S8535, ATMEGA8, ATMEGA16, ATMEGA64, ATMEGA128, ATMEGA168 xxxx meaning processor partnumber.
xxxxHDC.INC	Is a Global Device Handle Interrupts include file supplied by **Author** to possibility a dynamic interrupt handling such as interrupt real time Handdle address change and Interrupt cascading allowing that more one Handdle routine be used. The control of these handles is taken using below macros. **_SET_HANDDLE** to set a new Handle routine to a specific hardware interrupt. **_SAVE_HANDDLE** to save an actual Handle routine address. **_CALL_HANDDLE** to call a specific hardware interrupt. Author Handdle files actually support these devices AT90S1200, AT90S2313, AT90S2323, AT90S8515, AT90S8535, ATMEGA8, ATMEGA16, ATMEGA64, ATMEGA128, ATMEGA168 but take a look at one of these files quickly other devices are made, the only restriction is that file use the formatting **xxxxHDC.INC** x meaning device partnumber.

File	Description
Hardware_Definitions.Inc	Is a Global Definitions include file that contain all EQUATES, ROUTINES, etc, that is Hardware dependent.
Local_Definitions.Inc	Is a Global Definitions include file that contain all EQUATES,ROUTINES,FUNCTION AND MODULES used by MAIN program and no Hardware dependent routine must called directly by a MODULE if a OOP techniques is used, only FUNCTIONS can call a hardware dependent routine if OOP is used.

DEFS – Definitions

GLOBAL DEFINITIONS (GLOBDEFS.INC FILE)

Description

A GLOBAL DEFINITIONS include is the first include file in any program of **Author Tecnology.** This file contains all necessary Constants Equates and General Macros definitions to standardize code, increasing software reliability and turning it clear and reducing time for coding when working in group. **This file must be always included and must be a first include.**

	Accumulators registers	
Name	Register	Description
Acc	R24	8bits Accumulator Low
AccH	R25	8bits Accumulator High
AccT	R16	8bits Temporary Accumulator Low
AccTH	R17	8bits Temporary Accumulator High
Temp	R18	8bits used in short time like Temporary low register
TempH	R19	8bits used in short time like Temporary high register
Suffix AW Or AccH:Acc	R25:R24	16bits Accumulator
Suffix AWT Or AccTH:AccT	R17:R16	16bits Temporary Accumulator
Suffix AL Or AccTH:AccT:AccH:AccTH	R17:r16:r25:r24	32bits Accumulator

Global useful constants		
Name	**Decimal Value**	**Hexadecimal Value**
_ON	0XFF	255
_OFF	0X00	0
_EVEN	0XFF	255
_ODD	0XFF	255
_NONE	0X00	0
_LEFT	0X01	1
_RIGHT	0X02	2
_UP	0X03	3
_DOWN	0X04	4
_PRESSED	0X05	5
_RELEASED	0X06	6
_TIMEOUT	0X07	7
_YES	0XFF	255
_NO	0X00	0
_OK	0XFF	255
_NOTOK	0X00	0

Global ASCII Codes			
Name	**Decimal Value**	**Hexadecimal Value**	**meaning**
_NULL	0X00	0	Null Char
_BS	0X08	8	Backspace
_TAB	0X09	9	Tab(Tabulation)
_CR	0X0D	13	Carriage Return
_LF	0X0A	10	Line Feed
_NC	0XFF	255	Null Char(special use)
_ASCII_NULL	0X00	0	Null
_ASCII_SOH	0X01	1	Start of Heading
_ASCII_STX	0X02	2	Start of Text
_ASCII_ETX	0X03	3	End of Text
_ASCII_EOT	0X04	4	End of Transmission
_ASCII_ENQ	0X05	5	Enquiry
_ASCII_ACK	0X06	6	Acknowledge
_ASCII_BEL	0X07	7	Bell - Caused teletype machines to ring a bell. Causes a beep
_ASCII_BS	0X08	8	Backspace - Moves the cursor (or print head) move backwards (left)
_ASCII_TAB	0X09	9	Horizontal tab - Moves the cursor (or print head) right to the next
_ASCII_LF	0X0A	10	NL line feed, new line - Moves the cursor (or print head) to a new
_ASCII_VT	0X0B	11	vertical tab
_ASCII_FF	0X0C	12	Form feed - Advances paper to the

			top of the next page
_ASCII_CR	0X0D	13	Carriage return - Moves the cursor all the way to the left
_ASCII_SO	0X0E	14	Shift out - Switches output device to alternate character set.
_ASCII_SI	0X0F	15	shift in - Switches output device back to default character set.
_ASCII_DLE	0X10	16	Data link escape
_ASCII_DC1	0X11	17	Device control 1
_ASCII_DC2	0X12	18	Device control 2
_ASCII_DC3	0X13	19	Device control 3
_ASCII_DC4	0X14	20	Device control 4
_ASCII_NAK	0X15	21	Negative acknowledge
_ASCII_SYN	0X16	22	S ynchronous idle
_ASCII_ETB	0X17	23	End of transmission block - Not the same as EOT
_ASCII_CAN	0X18	24	Cancel
_ASCII_EM	0X19	25	End of medium
_ASCII_SUB	0X1A	26	Substitute
_ASCII_ESC	0X1B	27	Escape
_ASCII_FS	0X1C	28	File separator
_ASCII_GS	0X1D	29	Group separator
_ASCII_RS	0X1E	30	Record separator
_ASCII_US	0X1F	31	Unit separator
_ASCII_SPACE	0X20	32	Space
_ASCII_SP	0X20	32	Space
_ASCII_DEL	0X7F	127	Delete

Global Math constants		
Name	**Value**	**Description**
_LONG	4	Size of Long variable
_INTEGER	2	Size of Integer variable
_WORD	2	Size of Word variable
_BYTE	1	Size of byte variable

Global character display definitions for HD44780 chip constants		
Name	**Value**	**Description**
_DISP_LINE_1	0	1 Row display
_DISP_LINE_2	1	2 Row display
_DISP_FONT_5X8	0	5x8 Font Size
_DISP_FONT_5X10	1	5x10 Font Size
_DISP_4BITS	0	4 bits interface
_DISP_8BITS	1	8 bits interface

Wave constants		
Name	**Value**	**Description**
_WAVE_DAC_8	8	8 Bits wave DAC
_WAVE_DAC_16	16	16 Bits wave DAC
_WAVE_DAC_24	24	24 Bits wave DAC
_WAVE_DAC_32	32	32 Bits wave DAC
_WAVE_FS_5500	5500	5500 samples/second
_WAVE_FS_6000	6000	6000 samples/second
_WAVE_FS_8000	8000	8000 samples/second
_WAVE_FS_11025	11025	11025 samples/second
_WAVE_FS_22050	22050	22050 samples/second
_WAVE_FS_32000	32000	32000 samples/second
_WAVE_FS_44100	44100	44100 samples/second
_WAVE_SOURCE_FLASH	1	Wave into AVR Flash
_WAVE_SOURCE_SRAM	2	Wave into AVR SRAM
_WAVE_SOURCE_DEVICE	3	Wave Into DEVICE
_WAVE_STATUS_PLAYING	1	Wave Status playing
_WAVE_STATUS_STOPED	2	Wave Status Stoped
_WAVE_STATUS_END	3	Wave Status End

Sram boot type constants		
Name	**Value**	**Description**
_SRAM_NOT_CLEAR	0	Not clear SRAM during reset
_SRAM_CLEAR	1	Clear SRAM during reset

Prescaler constants for normal AVR chips		
Name	**Value**	**Description**
_TIMER_STOP	0B00000000	Timer stop
_TIMER_DIV_1	0B00000001	Timer prescaler divisor by 1
_TIMER_DIV_8	0B00000010	Timer prescaler divisor by 8
_TIMER_DIV_64	0B00000011	Timer prescaler divisor by 64
_TIMER_DIV_256	0B00000100	Timer prescaler divisor by 256
_TIMER_DIV_1024	0B00000101	Timer prescaler divisor by 1024
_TIMER_FALL	0B00000110	Timer Fall
_TIMER_RISE	0B00000111	Timer Rise

Prescaler constants for ATMEGA128 TIMER 0		
Name	**Value**	**Description**
_TIMERM0_STOP	0B00000000	Timer stop
_TIMERM0_DIV_1	0B00000001	Timer prescaler divisor by 1
_TIMERM0_DIV_8	0B00000010	Timer prescaler divisor by 8
_TIMERM0_DIV_32	0B00000011	Timer prescaler divisor by 32
_TIMERM0_DIV_64	0B00000100	Timer prescaler divisor by 64
_TIMERM0_DIV_128	0B00000101	Timer prescaler divisor by 128
_TIMERM0_DIV_256	0B00000110	Timer prescaler divisor by 256
_TIMERM0_DIV_1024	0B00000111	Timer prescaler divisor by 1024

Prescaler constants for ATMEGA128 TIMER 2		
Name	**Value**	**Description**
_TIMERM2_STOP	0B00000000	Timer stop
_TIMERM2_DIV_1	0B00000001	Timer prescaler divisor by 1
_TIMERM2_DIV_8	0B00000010	Timer prescaler divisor by 8
_TIMERM2_DIV_32	0B00000011	Timer prescaler divisor by 32
_TIMERM2_DIV_256	0B00000100	Timer prescaler divisor by 256
_TIMERM2_DIV_1024	0B00000101	Timer prescaler divisor by 1024
_TIMERM2_FALL	0B00000110	Timer prescaler divisor by FALL
_TIMERM2_RISE	0B00000111	Timer prescaler divisor by RISE

Prescaler constants for ATMEGA128 TIMER 1 & 3		
Name	**Value**	**Description**
_TIMERM13_STOP	0B00000000	Timer stop
_TIMERM13_DIV_1	0B00000001	Timer prescaler divisor by 1
_TIMERM13_DIV_8	0B00000010	Timer prescaler divisor by 8
_TIMERM13_DIV_64	0B00000011	Timer prescaler divisor by 64
_TIMERM13_DIV_256	0B00000100	Timer prescaler divisor by 256
_TIMERM13_DIV_1024	0B00000101	Timer prescaler divisor by 1024
_TIMERM13_FALL	0B00000110	Timer prescaler divisor by FALL
_TIMERM13_RISE	0B00000111	Timer prescaler divisor by RISE

Prescaler constants for ATMEGA128 ADC		
Name	**Value**	**Description**
_TIMERMADC_DIV_2	0B00000000	Timer prescaler divisor by 2
_TIMERMADC_DIV_4	0B00000001	Timer prescaler divisor by 4
_TIMERMADC_DIV_8	0B00000010	Timer prescaler divisor by 8
_TIMERMADC_DIV_16	0B00000011	Timer prescaler divisor by 16
_TIMERMADC_DIV_32	0B00000100	Timer prescaler divisor by 32
_TIMERMADC_DIV_64	0B00000101	Timer prescaler divisor by 64
_TIMERMADC_DIV_128	0B00000110	Timer prescaler divisor by 128

Interrupt pins sensing types constants		
Name	**Value**	**Description**
_LOW_LEVEL	0	Level sensing interrupt
_FALLING_EDGE	2	Falling edge sensing interrupt
_RISING_EDGE	3	Rising edge sensing interrupt

Timers Definitions constants		
Name	**Value**	**Description**
_TIMER_0	0	To assign reference to timer 0
_TIMER_1	1	To assign reference to timer 1
_TIMER_2	2	To assign reference to timer 2
_TIMER_3	3	To assign reference to timer 3

Interrupts Sources definitions constants		
Name	**Value**	**Description**
_EXTERNAL_0	0	To assign reference to external interrupt 0
_EXTERNAL_1	1	To assign reference to external interrupt 1
_EXTERNAL_2	2	To assign reference to external interrupt 2
_EXTERNAL_3	3	To assign reference to external interrupt 3
_EXTERNAL_4	4	To assign reference to external interrupt 4
_EXTERNAL_5	5	To assign reference to external interrupt 5
_EXTERNAL_6	6	To assign reference to external interrupt 6
_EXTERNAL_7	7	To assign reference to external interrupt 7

Communications definitions constants		
Name	**Value**	**Description**
_COMM0	0	To assign reference to COMM0
_COMM1	1	To assign reference to COMM1
_COMM2	2	To assign reference to COMM2
_COMM3	3	To assign reference to COMM3

MACRO	**_SET_HANDLE**
Function	Set Handle address routine
Example	Set Timer0 overflow interrupt to jump to ADDRESS_ROUTINE _SET_HANDLE _HDC_OVF0_VECT,ADDRESS_ROUTINE
Observation	Interrupt are disabled during save and register **AccH:Acc** is used

MACRO	_SAVE_HANDLE
Function	Save Handle address into SRAM
Example	Save Timer0 overflow interrupt address into SRAM_ADDRESS _SAVE_HANDLE _HDC_OVF0_VECT,SRAM_ADDRESS
Observation	Interrupt are disabled during save and register **AccH:Acc** is used

MACRO	_CALL_HANDLE
Function	Call Handle routine pointed by SRAM address
Example	Call Timer0 overflow interrupt _CALL_HANDLE _HDC_OVF0_VECT or _CALL_HANDLE SRAM_ADDRESS
Observation	**Z** register used to hold a address to be call

MACRO	LDIAW
Function	Load immediate value into **AccH:Acc**
Example	Load immediate **AccH:Acc** with 1500 LDIAW 1500

MACRO	LDIAWT
Function	Load immediate value into **AccTH:AccT**
Example	Load immediate **AccTH:AccT** with 1500 LDIAWT 1500

MACRO	LDIAL
Function	Load immediate value into **AccTH:AccT:AccH:Acc**
Example	Load immediate **AccTH:AccT:AccH:Acc** with 12345000 LDIAL 12345000

MACRO	LDIX
Function	Load immediate value into **X**
Example	Load immediate **X** with 1234 LDIX 1234

MACRO	LDIY
Function	Load immediate value into **Y**
Example	Load immediate **Y** with 1234 LDIY 1234

MACRO	LDIZ
Function	Load immediate value into Z
Example	Load immediate Z with 1234 LDIZ 1234

MACRO	CLRW
Function	Clear registers **X**,**Y** or **Z**
Example	Clear **X,Y and Z** CLRW X CLRW Y CLRW Z

MACRO	LDIW
Function	Load immediate registers **X**,**Y** or **Z**

Example	Load immediate **X**=123 **Y**=456 **Z**=789
	
	LDIW X,123
	LDIW Y,456
	LDIW Z,789

MACRO	LDSAW
Function	Load 16Bits **AccH:Acc** with memory contents position
Example	Load **AccH:Acc** with memory contents SRAM_POS LDSAW SRAM_POS

MACRO	LDSAWT

Function	Load 16Bits **AccTH:AccT** with memory contents position
Example	Load **AccTH:AccT** with memory contents SRAM_POS LDSAWT SRAM_POS

MACRO	**LDSAL**
Function	Load 32Bits **AccTH:AccT:AccH:Acc** with memory contents position
Example	Load **AccTH:AccT:AccH:Acc** with memory contents of SRAM_POS LDSAL SRAM_POS

MACRO	**LDDAW**
Function	Load 16Bits **AccH:Acc** with using **X**,**Y** or **Z** as base + index
Example	Load **AccH:Acc** with memory contents pointed by X+2 LDDAW X,2

MACRO	**LDDAWT**
Function	Load 16Bits **AccTH:AccT** with using **X**,**Y** or **Z** as base + index
Example	Load **AccTH:AccT** with memory contents pointed by X+2 LDDAWT X,2

MACRO	**LDSW**
Function	Load 16Bits **X**,**Y** or **Z** with memory position
Example	Load **Z with memory contents position SRAM_POS** LDSW Z,SRAM_POS

MACRO	LDDW
Function	Load 16Bits **X,Y or Z** with using **X,Y** or **Z** as base + index
Example	Load **Y** with memory contents pointed by X+2 LDDW Y,X,2

MACRO	STSAW
Function	Store 16Bits **AccH:Acc** into memory position
Example	Store **AccH:Acc** into memory SRAM_POS STSAW SRAM_POS

MACRO	STSAWT
Function	Store 16Bits **AccTH:AccT** into memory position
Example	Store **AccTH:AccT** into memory SRAM_POS STSAWT SRAM_POS

MACRO	STSAL
Function	Store 32Bits **AccTH:AccT:AccH:Acc** into memory position
Example	Store **AccTH:AccT:AccH:Acc** into memory SRAM_POS STSAL SRAM_POS

MACRO	STSW
Function	Store 16Bits **X**,**Y** or **Z** into memory position
Example	Store **Z into memory position SRAM_POS** STSW SRAM_POS,Z

MACRO	ADDI
Function	Add immediate to register
Example	Add immediate value 5 to register Acc ADDI Acc,5

MACRO	ADCI
Function	Add immediate with carry to register
Example	Add immediate with carry value 5 to register Acc ADCI Acc,5

MACRO	SUBIAW
Function	Subtract 16Bits **AccH:Acc** with immediate value
Example	Subtract **AccH:Acc** with 45 SUBIAW 45

MACRO	SUBIAWT
Function	Subtract 16Bits **AccTH:AccT** with immediate value
Example	Subtract **AccTH:AccT** with 45 SUBIAWT 45

MACRO	SUBIAL
Function	Subtract 32Bits **AccTH:AccT:AccH:Acc** with immediate value
Example	Subtract **AccTH:AccT:AccH:Acc** with 12345678 SUBIAL 12345678

MACRO	ADDIAW
Function	Add 16Bits **AccH:Acc** with immediate value
Example	Add **AccH:Acc** with 1234 ADDIAW 1234

MACRO	ADDIAWT
Function	Add 16Bits **AccTH:AccT** with immediate value
Example	Add **AccTH:AccT** with 1234 ADDIAWT 1234

MACRO	ADDIAL
Function	Add 32Bits **AccTH:AccT:AccH:Acc** with immediate value
Example	Add **AccTH:AccT:AccH:Acc** with 12345678 ADDIAL 12345678

MACRO	SUBIW
Function	Subtract 16Bits **X**,**Y** or **Z** with immediate value
Example	Subtract **Y** with 78 SUBIW Y,78

MACRO	LSRW
Function	Logical shift right word register **X**,**Y**,**Z**
Example	Logical shift right word X LSRW X

MACRO	ASRW
Function	Arithmetic shift right word register **X**,**Y**,**Z**
Example	Arithmetic shift right word X ASRW X

MACRO	LSLAW
Function	Logical shift left 16Bits **AccH:Acc**
Example	Logical shift left **AccH:Acc** LSLAW

MACRO	LSLAWT
Function	Logical shift left 16Bits **AccTH:AccT**
Example	Logical shift left **AccTH:AccT** LSLAWT

MACRO	LSRAW
Function	Logical shift right 16Bits **AccH:Acc**
Example	Logical shift right **AccH:Acc** LSRAW

MACRO	LSRAWT
Function	Logical shift right 16Bits **AccTH:AccT**
Example	Logical shift right **AccTH:AccT** LSRAWT

MACRO	ASRAW
Function	Arithmetic shift right 16Bits **AccH:Acc**
Example	Arithmetic shift right **AccH:Acc** ASRAW

MACRO	ASRAWT
Function	Arithmetic shift right 16Bits **AccHT:AccT**
Example	Arithmetic shift right **AccTH:AccT** ASRAWT

MACRO	SUBW
Function	Subtract **X**,**Y**,**Z** from **X**,**Y**,**Z**
Example	Subtract **X** from **Z** SUBW **X**,**Z**

MACRO	ADDIW
Function	Add **X**,**Y**,**Z** with immediate value
Example	Add **Z** with 1234 ADDIW Z,1234

MACRO	ADDW
Function	Add X,Y,Z with X,Y,Z
Example	Add X with Z ADDW X,Z

MACRO	CPIW
Function	Compare X,Y,Z with immediate value
Example	Compare X with 7 CPIW X,7
Observation	Temp destroyed

MACRO	CPW
Function	Compare X,Y,Z with X,Y,Z
Example	Compare X with Z CPW X,Z

MACRO	CPIAW
Function	Compare 16Bits AccH:Acc with immediate value
Example	Compare 16Bits AccH:Acc with 127 CPIAW 127
Observation	Temp destroyed

MACRO	CPIAL
Function	Compare 32Bits AccTH:AccT:AccH:Acc with immediate value
Example	Compare AccTH:AccT:AccH:Acc with 12345678 CPIAL 12345678
Observation	Temp destroyed

MACRO	PUSHAW
Function	Push into stack 16Bits AccH:Acc
Example	Push AccH:Acc PUSHAW

MACRO	PUSHAWT
Function	Push into stack 16Bits AccTH:AccT

Example	Push **AccTH:AccT** PUSHAWT

MACRO	**PUSHTEMPW**
Function	Push into stack 16Bits **TempH:Temp**
Example	Push **TempH:Temp** PUSHTEMPW

MACRO	**PUSHW**
Function	Push into stack 16Bits **X,Y,Z**
Example	Push **Z** PUSHW **Z**

MACRO	**PUSHW**
Function	Push into stack 16Bits **X,Y,Z**
Example	Push **Z** PUSHW **Z**

MACRO	**POPAW**
Function	Pop from stack 16Bits **AccH:Acc**
Example	Pop **AccH:Acc** from stack POPAW

MACRO	**POPAWT**
Function	Pop from stack 16Bits **AccTH:AccT**
Example	Pop **AccTH:AccT** from stack POPAWT

MACRO	POPTEMPW
Function	Pop from stack 16Bits **TempH:Temp**
Example	Pop **TempH:Temp** from stack POPTEMPW

MACRO	POPW
Function	Pop from stack 16Bits **X**,**Y**,**Z**
Example	Pop **Z** from stack POPW Z

MACRO	LBRNE
Function	Long Branch not equal(no limit of +-2k)
Example	Branch to Address if not Egual LBRNE Address

MACRO	LBREQ
Function	Long Branch equal(no limit of +-2k)
Example	Branch to Address if Egual LBREQ Address

MACRO	LBRCS
Function	Long Branch if Carry bit Set(no limit of +-2k)
Example	Branch to Address if not CY=1 LBRCS Address

MACRO	LBRCC
Function	Long Branch if Carry bit Clear(no limit of +-2k)
Example	Branch to Address if not CY=0 LBRCC Address

MACRO	LBRLT
Function	Long Branch if less than signed(no limit of +-2k)
Example	Branch to Address if less than LBRLT Address

MACRO	LBRMI
Function	Long Branch if Minus(no limit of +-2k)
Example	Branch to Address if minus LBRMI Address

MACRO	LBRPO
Function	Long Branch if Positive(no limit of +-2k)
Example	Branch to Address if Positive LBRPO Address

MACRO	LBRGE
Function	Long Branch if Great or Igual signed(no limit of +-2k)
Example	Branch to Address if Great or Igual LBRGE Address

MACRO	LBRSH
Function	Long Branch if Same or High unsigned(no limit of +-2k)
Example	Branch to Address if Same or High LBRSH Address

MACRO	LBRLO
Function	Long Branch if Lower(no limit of +-2k)
Example	Branch to Address if Lower LBRLO Address

MACRO	_M_PUSH_LOWER_REGS
Function	Push into stack registers R0..R15
Example	Pushing registers R0..R15 _M_PUSH_LOWER_REGS

MACRO	_M_PUSH_UPPER_REGS
Function	Push into stack registers R16..R31
Example	Pushing registers R16..R31 _M_PUSH_UPPER_REGS

MACRO	_M_PUSH_ALL_REGS
Function	Push into stack registers R0..R31
Example	Pushing registers R0..R31 _M_PUSH_ALL_REGS

MACRO	_M_POP_LOWER_REGS
Function	Pop from stack registers R0..R15
Example	Poping registers R0..R15 _M_POP_LOWER_REGS

MACRO	_M_POP_UPPER_REGS
Function	Pop into stack registers R16..R31
Example	Poping registers R16..R31 _M_POP_UPPER_REGS

MACRO	_M_POP_ALL_REGS
Function	Pop into stack registers R0..R31
Example	Poping registers R0..R31 _M_POP_ALL_REGS

AVR Family chips names definitions and Interrupts Handdles for following devices 1200,2313,2323,8515,8535,M8,M16,M64,M128,M162

Description

Each AVR Microcontroller device has a lot of internal register, these registers are used to set mode of operation of internal features like USART,PWM,TIMER,GPIO,I2C,ADC etc, to facility use of these register names is assigned to them, the files with this names has the following format XXXXDEF.INC where XXXX is a device number, below a list of Definitions files used by **Author**;

1200DEF.INC
2313DEF.INC
8515DEF.INC
8535DEF.INC
M8.INC
M16.INC
M64.INC
M128.INC
M162.INC

Furthermore each device has a interrupt handle file that allow dynamic handling of interrupts like assigned in run time a routine that process this interrupt or cascading then. Below a tables describing handles of each device.

(AT90S2313) 2313HDC.INC HANDLE FILE	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_ICP1_VECT	Timer1 capture interrupt handle
_HDC_OC1_VECT	Timer1 compare interrupt handle
_HDC_OVF0_VECT	Timer0 Overflow interrupt handle
_HDC_URXC_VECT	UART RX complete interrupt handle
_HDC_UDRE_VECT	UDR Empty interrupt handle
_HDC_UTXC_VECT	UART TX complete interrupt handle
_HDC_ACI_VECT	Analog comparator interrupt handle

(AT90S2323) 2323HDC.INC HANDLE FILE	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_OVF0_VECT	Timer0 Overflow interrupt handle

(AT90S8515) 8515HDC.INC HANDLE FILE	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_ICP1_VECT	Timer1 capture interrupt handle
_HDC_OC1A_VECT	Timer1 compare math A interrupt handle
_HDC_OC1B_VECT	Timer1 compare math B interrupt handle
_HDC_OVF1_VECT	Timer1 Overflow interrupt handle
_HDC_OVF0_VECT	Timer0 Overflow interrupt handle
_HDC_SPI_VECT	SPI Serial transfer interrupt handdle
_HDC_URXC_VECT	UART RX complete interrupt handle
_HDC_UDRE_VECT	UDR Empty interrupt handle
_HDC_UTXC_VECT	UART TX complete interrupt handle
_HDC_ACI_VECT	Analog comparator interrupt handle

(AT90S8535) 8535HDC.INC HANDLE FILE	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_TOC2_VECT	Timer2 compare math interrupt handle
_HDC_OVF2_VECT	Timer2 overflow interrupt handle
_HDC_ICP1_VECT	Timer1 capture interrupt handle
_HDC_OC1A_VECT	Timer1 compare math A interrupt handle
_HDC_OC1B_VECT	Timer0 compare math B interrupt handle
_HDC_OVF1_VECT	Timer1 overflow interrupt handdle
_HDC_OVF0_VECT	Timer0 overflow interrupt handle
_HDC_SPI_VECT	SPI serial transfer complete interrupt handle
_HDC_URXC_VECT	UART RX complete interrupt handle
_HDC_UDRE_VECT	UDR Empty interrupt handle
_HDC_UTXC_VECT	UART TX complete interrupt handle
_HDC_ADC_VECT	ADC conversion complete interrupt handle
_HDC_EERDY_VECT	EEPROM Ready interrupt handle
_HDC_ACI_VECT	Analog comparator interrupt handle

(ATMEGA8) M8HDC.INC HANDLE FILE	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_OC2_VECT	Timer2 compare math interrupt handle
_HDC_OVF2_VECT	Timer2 overflow interrupt handle
_HDC_ICP1_VECT	Timer1 capture interrupt handle
_HDC_OC1A_VECT	Timer1 compare math A interrupt handle
_HDC_OC1B_VECT	Timer0 compare math B interrupt handle
_HDC_OVF1_VECT	Timer1 overflow interrupt handdle
_HDC_OVF0_VECT	Timer0 overflow interrupt handle
_HDC_SPI_VECT	SPI serial transfer complete interrupt handle
_HDC_URXC_VECT	UART RX complete interrupt handle
_HDC_UDRE_VECT	UDR Empty interrupt handle
_HDC_UTXC_VECT	UART TX complete interrupt handle
_HDC_ADCC_VECT	ADC conversion complete interrupt handle
_HDC_ERDY_VECT	EEPROM Ready interrupt handle
_HDC_ACI_VECT	Analog comparator interrupt handle
_HDC_TWI_VECT	Two-wire serial interface interrupt handle
_HDC_SPMR_VECT	Store Program Memory Ready interrupt handle

(ATMEGA16) M16HDC.INC HANDLE FILE	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_OC2_VECT	Timer2 compare math interrupt handle
_HDC_OVF2_VECT	Timer2 overflow interrupt handle
_HDC_ICP1_VECT	Timer1 capture interrupt handle
_HDC_OC1A_VECT	Timer1 compare math A interrupt handle
_HDC_OC1B_VECT	Timer0 compare math B interrupt handle
_HDC_OVF1_VECT	Timer1 overflow interrupt handle
_HDC_OVF0_VECT	Timer0 overflow interrupt handle
_HDC_SPI_VECT	SPI serial transfer complete interrupt handle
_HDC_URXC0_VECT	UART RX complete interrupt handle
_HDC_UDRE0_VECT	UDR Empty interrupt handle
_HDC_UTXC0_VECT	UART TX complete interrupt handle
_HDC_ADCC_VECT	ADC conversion complete interrupt handle
_HDC_ERDY_VECT	EEPROM Ready interrupt handle
_HDC_ACI_VECT	Analog comparator interrupt handle
_HDC_TWI_VECT	Two-wire serial interface interrupt handle
_HDC_INT2_VECT	External Interrupt 2
_HDC_OCO_VECT	Timer0 compare math interrupt handle
_HDC_SPMR_VECT	Store Program Memory Ready interrupt handle

(ATMEGA64) M64HDC.INC HANDLE FILE Model A (SRAM Start 0x100)	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_INT2_VECT	External Interrupt IRQ2
_HDC_INT3_VECT	External Interrupt IRQ3
_HDC_INT4_VECT	External Interrupt IRQ4
_HDC_INT5_VECT	External Interrupt IRQ5
_HDC_INT6_VECT	External Interrupt IRQ6
_HDC_INT7_VECT	External Interrupt IRQ7
_HDC_OC2_VECT	Timer2 compare math interrupt handle
_HDC_OVF2_VECT	Timer2 overflow interrupt handle
_HDC_ICP1_VECT	Timer1 capture interrupt handle
_HDC_OC1A_VECT	Timer1 compare math A interrupt handle
_HDC_OC1B_VECT	Timer1 compare math B interrupt handle
_HDC_OVF1VECT	Timer1 overflow interrupt handle
_HDC_OC0_VECT	Timer0 compare math interrupt handle
_HDC_OVF0_VECT	Timer0 overflow interrupt handle
_HDC_SPI_VECT	SPI Serial transfer complete interrupt handle
_HDC_URXC0_VECT	USART0 Rx complete interrupt handle
_HDC_UDRE0_VECT	USART0 Data register empty interrupt handle
_HDC_UTXC0_VECT	USART0 Tx complete interrupt handle
_HDC_ADCC_VECT	ADC conversion complete interrupt handle
_HDC_ERDY_VECT	EEPROM ready interrupt handle
_HDC_ACI_VECT	Analog comparator interrupt handle
_HDC_OC1C_VECT	Timer1 compare math C interrupt handle
_HDC_ICP3_VECT	Timer3 capture interrupt handle
_HDC_OC3A_VECT	Timer3 compare math A interrupt handle
_HDC_OC3B_VECT	Timer3 compare math B interrupt handle
_HDC_OC3C_VECT	Timer3 compare math C interrupt handle
_HDC_OVF3_VECT	Timer3 overflow interrupt handle
_HDC_URXC0_VECT	USART1 Rx complete interrupt handle
_HDC_UDRE0_VECT	USART1 Data register empty interrupt handle
_HDC_UTXC0_VECT	USART1 Tx complete interrupt handle
_HDC_TWI_VECT	Two-wire serial interface interrupt handle
_HDC_SPMR_VECT	Store program memory Ready interrupt handle

(ATMEGA128) M128HDC.INC HANDLE FILE Model A (SRAM Start 0x100)	
HANDLES	Description
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_INT2_VECT	External Interrupt IRQ2
_HDC_INT3_VECT	External Interrupt IRQ3
_HDC_INT4_VECT	External Interrupt IRQ4
_HDC_INT5_VECT	External Interrupt IRQ5
_HDC_INT6_VECT	External Interrupt IRQ6
_HDC_INT7_VECT	External Interrupt IRQ7
_HDC_OC2_VECT	Timer2 compare math interrupt handle
_HDC_OVF2_VECT	Timer2 overflow interrupt handle
_HDC_ICP1_VECT	Timer1 capture interrupt handle
_HDC_OC1A_VECT	Timer1 compare math A interrupt handle
_HDC_OC1B_VECT	Timer1 compare math B interrupt handle
_HDC_OVF1VECT	Timer1 overflow interrupt handle
_HDC_OC0_VECT	Timer0 compare math interrupt handle
_HDC_OVF0_VECT	Timer0 overflow interrupt handle
_HDC_SPI_VECT	SPI Serial transfer complete interrupt handle
_HDC_URXC0_VECT	USART0 Rx complete interrupt handle
_HDC_UDRE0_VECT	USART0 Data register empty interrupt handle
_HDC_UTXC0_VECT	USART0 Tx complete interrupt handle
_HDC_ADCC_VECT	ADC conversion complete interrupt handle
_HDC_ERDY_VECT	EEPROM ready interrupt handle
_HDC_ACI_VECT	Analog comparator interrupt handle
_HDC_OC1C_VECT	Timer1 compare math C interrupt handle
_HDC_ICP3_VECT	Timer3 capture interrupt handle
_HDC_OC3A_VECT	Timer3 compare math A interrupt handle
_HDC_OC3B_VECT	Timer3 compare math B interrupt handle
_HDC_OC3C_VECT	Timer3 compare math C interrupt handle
_HDC_OVF3_VECT	Timer3 overflow interrupt handle
_HDC_URXC0_VECT	USART1 Rx complete interrupt handle
_HDC_UDRE0_VECT	USART1 Data register empty interrupt handle
_HDC_UTXC0_VECT	USART1 Tx complete interrupt handle
_HDC_TWI_VECT	Two-wire serial interface interrupt handle
_HDC_SPMR_VECT	Store program memory Ready interrupt handle

(ATMEGA162) M162HDC.INC HANDLE FILE Model A (SRAM Start 0x100)	
HANDLES	**Description**
_HDC_INT0_VECT	External Interrupt IRQ0
_HDC_INT1_VECT	External Interrupt IRQ1
_HDC_INT2_VECT	External Interrupt IRQ2
_HDC_TIMER3_CAPT_VECT	Timer3 capture interrupt handle
_HDC_TIMER3_COMPA_VECT	Timer3 compare math A interrupt handle
_HDC_TIMER3_COMPB_VECT	Timer3 compare math B interrupt handle
_HDC_TIMER3_OVF_VECT	Timer3 overflow interrupt handle
_HDC_TIMER2_COMP_VECT	Timer2 compare math interrupt handle
_HDC_TIMER2_OVF_VECT	Timer2 overflow interrupt handle
_HDC_TIMER1_CAPT_VECT	Timer1 capture interrupt handle
_HDC_TIMER1_COMPA_VECT	Timer1 compare math A interrupt handle
_HDC_TIMER1_COMPB_VECT	Timer1 compare math B interrupt handle
_HDC_TIMER1_OVF_VECT	Timer1 overflow interrupt handle
_HDC_TIMER0_COMP_VECT	Timer0 compare math interrupt handle
_HDC_TIMER0_OVF_VECT	Timer0 overflow interrupt handle
_HDC_SPI_VECT	SPI Serial transfer interrupt handle
_HDC_USART0_RXC_VECT	USART0 Rx complete interrupt handle
_HDC_USART1_RXC_VECT	USART1 Rx complete interrupt handle
_HDC_USART0_UDRE_VECT	USART0 Data register empty interrupt handle
_HDC_USART1_UDRE_VECT	USART1 Date register empty interrupt handle
_HDC_USART0_TXC_VECT	USART0 Tx complete interrupt handle
_HDC_USART1_TXC_VECT	USART1 Tx complete interrupt handle
_HDC_EE_RDY_VECT	Timer3 compare math A interrupt handle
_HDC_ANA_COMP_VECT	Timer3 compare math B interrupt handle
_HDC_SPM_RDY_VECT	Timer3 compare math C interrupt handle

MATH DEFINITIONS (MathCons.Inc File)

Description

Some constants, SRAM variables and register definitions used for math computation using integer or float types. **This must be included if conversion to string is required or use float type variables.**

Double Float Point Memory Formating							
Mantissa						Signal	Exponent
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

- **Byte 0 – LSB(Less significant Byte)**
- **Byte 6 – Bit 7 mantissa signal 0 positive 1 negative**
- **Byte 7 – Exponent – Bit 7 exponent signal 0 positive 1 negative**
-

Double Float Point BCD Memory Formating							
Mantissa							Exponent
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
D1-2	D3-4	D5-6	D7-8	D9-10	D11-12	D13-14	E12+ES+ES

- **Byte 0 to byte 6 – BCD digits 1 to 14**
- **Byte 7 – E12 exponent digits 1 and 2**
  Bit 6 exponent signal 0 positive 1 negative
  Bit 7 mantissa signal 0 positive 1 negative

Global Math constants		
Name	**Value**	**Description**
_DF_STR_BUF_SIZE	1+1+16+1+1+2=22	Space used for numeric to string conversions
_FDOUBLE_STACK_SIZE	8 (default)	Default size of float point stack
_FDOUBLE	8	Size of float point double variable
_FSINGLE	4	Size of float point single variable
_FBCD	8	Size of BCD(Binary Codec Decimal) variable
_FSTRING	_DF_STR_BUF_SIZE	Float string numbers
_FBIAS	0X81	Bias used to simplify float point computations
_FPOK	0	Float Point operation Ok
_FEPOVER	1	Float Point error overflow
_FEUNDER	2	Float Point error underflow
_FEDIV0	3	Float point error division by zero
_FEILLEG	4	Float point error illegal operator

Double Float point Operator 1		
Name	**Register**	**Description**
_op1_0	R0	1st operator mantissa byte 1
_op1_1	R1	1st operator mantissa byte 2
_op1_2	R2	1st operator mantissa byte 3
_op1_3	R3	1st operator mantissa byte 4
_op1_4	R4	1st operator mantissa byte 5
_op1_5	R5	1st operator mantissa byte 6
_op1_s	R6	1st operator mantissa byte 7 (signal)
_op1_e	R7	1st operator exponent byte 8

Double Float point Operator 2		
Name	**Register**	**Description**
_op2_0	R8	2st operator mantissa byte 1
_op2_1	R9	2st operator mantissa byte 2
_op2_2	R10	2st operator mantissa byte 3
_op2_3	R11	2st operator mantissa byte 4
_op2_4	R12	2st operator mantissa byte 5
_op2_5	R13	2st operator mantissa byte 6
_op2_s	R14	2st operator mantissa byte 7 (signal)
_op2_e	R15	2st operator exponent byte 8

Double Float point Accumulator		
Name	**Register**	**Description**
_op2_0	R18	2st operator mantissa byte 1(Temp)
_op2_1	R19	2st operator mantissa byte 2(TempH)
_op2_2	R20	2st operator mantissa byte 3
_op2_3	R21	2st operator mantissa byte 4
_op2_4	R22	2st operator mantissa byte 5
_op2_5	R23	2st operator mantissa byte 6
_op2_s	R26	2st operator mantissa byte 7 (signal) XL
_op2_e	R27	2st operator exponent byte 8 XH

Double Float Math SRAM Variables		
Name	**Size**	**Description**
_DF_MAC	_FDOUBLE	Mantissa accumulator
_DF_P10	_FDOUBLE	Power of 10
_DF_BCD	_FBCD	Float point codec BCD
_IS_SIZE	1	Define string output size of integer conversion. high nibble integer part size 0..15 low nibble decimal part size 0..15c
_DF_EAC	1	exponent accumulator
_DF_FMASK	1	string flags
_DF_MASK	1	BIT 0=1 + sign for positive numbers 1=1 use thousand separation char 2=1 separation char is (point) else (comma) 3=1 round result 0 no round result
_DF_FS	1	0X00 source is FLASH 0x01 source is SRAM
_DF_FRAC	1	this variable is used to say that FDIV is fractionary when value is 0xff
_DF_FSREG	1	hold a STATUS REGISTER flags of float point comparation

_DF_STR	**_DF_STR_BUF_SIZE**	**Used for string conversions**

MACRO	**_M_PUSH_MATH_VARIABLES**
Function	Save into stack all math variables
Example	_M_PUSH_MATH_VARIABLES

MACRO	**_M_POP_MATH_VARIABLES**
Function	Restore into stack all math variables
Example	_M_POP_MATH_VARIABLES

SRAM INIT (SRAM_INIT.Inc File)

Description

During power on, some times are required that all variables in SRAM must be cleared other times not. This include, control when SRAM is cleared of not and initialize normal float point division when MathConst.inc is used.

Use below code to clear SRAM during power on.

```
.EQU  _SRAM_BOOT_TYPE = _SRAM_CLEAR
```

Or below code to remainder SRAM state during power on.

```
.EQU  _SRAM_BOOT_TYPE = _SRAM_NOT_CLEAR
```

Those above equates must be a first equate in program

MATH

DFT – Discrete Fourier Transform

DFT_8BITS_64POINTS (DFT864V1.INC File)

Description

Discrete Fourier Transform (DFT) has a lot of application in digital signal process like Digital Filters, Frequency Finder, Frequency genlock, wave compress, DTMF recognition, Etc. This routines implements 64 points of DFT means that capable to separate 32 frequencies and get yours amplitudes, working with 8bits of data magnitude with 48db of signal noise ratio. One frequency index of DFT routine is computed using below follow equation:

The DFT for one point frequency is obtained using the next equation

$$v = \left(\sum_{i=0}^{63} dt(i).\sin\left(\frac{2\pi f i}{64}\right)\right) + \left(\sum_{i=0}^{63} dt(i).\cos\left(\frac{2\pi f i}{64}\right)\right)$$

Note that correct vectored sum is performed as follow

$$c = \sqrt{a^{2} + b^{2}}$$

but this method is more complex to calculate then to reduce time computation the equation below is used in place.

$$c = |a| + |b|$$

Doing this transformation a imprecision is introduced by a factor of $\sqrt{2}$ that is soothed by the dynamics of input signal $dt(i)$ and larger values of output function.

Implemented Functions

Name	_DFT_64B
Function	Compute one frequency magnitude using DFT.
Input values	**Acc** Frequency index 1..32
Output values	**AccH:Acc** Output frequency power Max value 2030 when signal at 0° Max value 2870 when signal at 45°
Destroy	Flags, R0..R13
Time	Average timing = 5258 clocks 4Mhz 1314us 8Mhz 657us 14.3Mhz 368us
Observations	Procedure to use then routine Fill _DFT_DATA_BUF with 64 data into SRAM at sample frequency FS, after this set **Acc** with index of frequency to get amplitude into **AccH:Acc**
Example	Suppose that FS=6000Hz FIndex0 =FS/64 <- Min Frequency FIndex31=FIndex0*32 <- Max Frequency FIndex0 = 93.75Hz ... FIndex31=3000.00 Now, if you need to obtain amplitude of Frequency 750Hz. ldi **Acc**,8 ;FIndex8=int(750/FIndex0) rcall _DDFTB ;After this point **AccH:Acc** ;have a amplitude of index ;frequency

Name	_DFT_DATA_CLEAR
Function	Clear data in DFB data buffer all igual 0
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Clear all data in DFT buffer rcall _DFT_DATA_CLEAR call _DFT_DATA_CLEAR (chips >=16k)

Name	***_DFT_BUFFER_FILL***
Function	Check if DFT data buffer is full
Input values	None
Output values	**Cy**=1 indicating buffer is full
Destroy	Flags
Time	----
Observations	----
Example	Check DFT buffer is full, after this call **CY**=1 if buffer full. rcall _DFT_BUFFER_FILL call _DFT_BUFFER_FILL (chips >=16k)

Name	***_DFT_DATA_ADD***
Function	Insert data in DFT data buffer
Input values	**Acc** data
Output values	**Cy**=1 indicating buffer is full
Destroy	Flags
Time	----
Observations	----
Example	Insert a new data in DFT data buffer Ldi **Acc**,15 rcall _DFT_DATA_ADD call _DFT_DATA_ADD (chips >=16k)

FILTERS

AVERAGE16 (AVERAGE16.INC File)

Description

Average16 circular filter is used to perform a low pass digital filter can be used to remove noise or high frequencies in a signal. Before use this routines set data buffer size as bellow:

```
.EQU  _FILTER_AVG16_SIZE = (SIZE)
```

First data buffer is shifted to accommodate a new data value as follow.

$$d(0) = d(1), d(1) = d(2), .. d(n-1) = d(n), d(n) = data$$

Then the output value of average filter is a average value of below equation

$$dout = \frac{d(0) + d(1) + d(2) + \cdots + d(n)}{n}$$

Implemented Functions

Name	***_FILTER_AVG16_SET_VALUE***
Function	Insert a new value into data buffer
Input values	**AccH:Acc** value
Output values	None
Destroy	None
Time	----
Observations	----
Example	Set a new value for average computation Ldiaw 1200 rcall _FILTER_AVG16_SET_VALUE call _FILTER_AVG16_SET_VALUE (chips >=16k)

Name	_FILTER_AVG16_GET_VALUE
Function	Get a output value from data buffer
Input values	None
Output values	AccH:Acc value
Destroy	None
Time	----
Observations	----
Example	Get a new value from data buffer, after this AccH:Acc has the value. rcall _FILTER_AVG16_GET_VALUE call _FILTER_AVG16_GET_VALUE (chips >=16k)

Name	_FILTER_AVG16_EXECUTE
Function	Shift the data buffer and compute new output value
Input values	None
Output values	AccH:Acc value Or use _FILTER_AVG16_GET_VALUE
Destroy	Flags,R0..R12
Time	----
Observations	----
Example	Compute new output value, after this AccH:Acc has the value of call _FILTER_AVG16_GET_VALUE rcall _FILTER_AVG16_EXECUTE call _FILTER_AVG16_EXECUTE (chips >=16k)

FLOAT DOUBLE

ACOS – ArcCosine (ACOS.INC File)

Description

Double float point ArcCosine is calculated using bellow equation.

$$\operatorname{acos}(x) = \frac{\pi}{2} - \operatorname{asin}(x)$$

Implemented Functions

Name	***_DFACOS***
Function	Compute Float Double ArcCosine
Input values	**Float Acc**
Output values	**Float Acc** **Acc** Exception code
Destroy	Flags,**AccH**,R0..R15
Time	----
Observations	----
Example	rcall _DFACOS call _DFACOS (chips >=16k)

ASIN – ArcSine (ASIN.INC File)

Description

Double float point ArcSine is calculated using bellow equation.

$$\operatorname{asin}(x) = atan\left(\frac{x}{\sqrt{1-x^2}}\right)$$

Implemented Functions

Name	**_DFASIN**
Function	Compute Float Double ArcSine
Input values	**Float Acc**
Output values	**Float Acc** **Acc** Exception code
Destroy	Flags,**AccH**,R0..R15
Time	----
Observations	----
Example	rcall _DFASIN call _DFASIN (chips >=16k)

ATAN – ArcTangent (ATAN.INC File)

Description

Double float point ArcTangent is computed using a _DFSERATN that compute a partial arctangent of value in range $[-\sqrt{2}+1, \sqrt{2}-1]$ then the below equation is used to obtain ArcTangent for full range of values.

$$\operatorname{atan}(x) = \begin{cases} seratn(x), if\ x < \sqrt{2}-1 \\ \frac{\pi}{2} - seratn\left(\frac{1}{x}\right), if\ x > \sqrt{2}+1 \\ \frac{\pi}{4} + seratn\left(\frac{x-1}{x+1}\right), if\ \sqrt{2}-1 < x < \sqrt{x}+1 \end{cases}$$

Implemented Functions

Name	**_DFATAN**
Function	Compute Float Double ArcTangent
Input values	**Float Acc**
Output values	**Float Acc** **Acc** Exception code
Destroy	Flags,**AccH**,R0..R15
Time	----
Observations	----
Example	rcall _DFATAN call _DFATAN (chips >=16k)

ADDSUB – Add and Subtraction (DFADDSUB.INC File)

Description

Double float point Addition and Subtraction.

Implemented Functions

Name	**_DFADD**
Function	Perform Float Double Addition
Input values	**Float Op1 1st operand** **Float Op2 2nd operand**
Output values	**Float Acc** **Acc** Exception code
Destroy	Flags,**AccH**,R0..R15
Time	----
Observations	Perform Float Acc=Float Op1+Float Op2
Example	rcall _DFADD call _DFADD (chips >=16k)

Name	**_DFSUB**
Function	Perform Float Double Subtraction
Input values	**Float Op1 1st operand** **Float Op2 2nd operand**
Output values	**Float Acc** **Acc** Exception code
Destroy	Flags,**AccH**,R0..R15
Time	----
Observations	Perform Float Acc=Float Op1-Float Op2
Example	rcall _DFSUB call _DFSUB (chips >=16k)

MULDIV – Multiply and divide (DEFMULDIV.INC File)

Description

Double float point Multiply and Divide.

Implemented Functions

Name	**_DFMUL**
Function	Perform Float Double Multiply
Input values	**Float Op1 1st operand** **Float Op2 2nd operand**
Output values	**Float Acc** **Acc** Exception code
Destroy	Flags,**AccH**,R0..R15
Time	----
Observations	Perform Float Acc=Float Op1*Float Op2
Example	rcall _DFMUL call _DFMUL (chips >=16k)

Name	**_DFDIV**
Function	Perform Float Double Divide
Input values	**Float Op1 1st operand** **Float Op2 2nd operand**
Output values	**Float Acc** **Acc** Exception code
Destroy	Flags,**AccH**,R0..R15
Time	----
Observations	Perform Float Acc=Float Op1/Float Op2
Example	rcall _DFDIV call _DFDIV (chips >=16k)

DFCPM – Float compare (DEFCPM.INC File)

Description

Compare two Double float point, after compare status flags will updated according compared values.

Implemented Functions

Name	**_DFCPOP1OP2**
Function	Perform Float Double Multiply
Input values	**Float Op1 1st operand** **Float Op2 2nd operand**
Output values	Status register _DF_FSREG with the copy of **SREG**
Destroy	**Acc**
Time	----
Observations	----
Example	rcall _DFCPOP1OP2 call _DFCPOP1OP2 (chips >=16k)

Below macros may be used to facility branch instruction after compare.

MACRO	**_DFJP_EQ**
Function	Branch to Address if compare is equal
Example	Jump to Address if Op1=Op2 _DFJP_EQ Address

MACRO	**_DFJP_NEQ**
Function	Branch to Address if compare is not equal
Example	Jump to Address if Op1<>Op2 _DFJP_NEQ Address

MACRO	**_DFJP_GT**
Function	Branch to Address if Op1>Op2
Example	Jump to Address if Op1>Op2 _DFJP_GT Address

MACRO	**_DFJP_LT**
Function	Branch to Address if Op1<Op2
Example	Jump to Address if Op1<Op2 _DFJP_LT Address

MACRO	**_DFJP_GTEQ**
Function	Branch to Address if Op1>=Op2
Example	Jump to Address if Op1>=Op2 _DFJP_GTEQ Address

MACRO	**_DFJP_LTEQ**
Function	Branch to Address if Op1<=Op2
Example	Jump to Address if Op1<=Op2 _DFJP_LTEQ Address

COSINE (COSINE.INC File)

Description

Compute Double float point Cosine using below equation

$$\cos(x) = \sin\left(x + \frac{\pi}{2}\right)$$

Implemented Functions

Name	_DFCOS
Function	Perform Float Double Cosine
Input values	**Float Acc**
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,R0..R15,AccH**
Time	----
Observations	----
Example	rcall _DFCOS call _DFCOS (chips >=16k)

SINE (SINE.INC File)

Description

Compute Double float point Sine using partial sine function _DFSERSIN using the below equation:

First reduce value=x to $2.\pi$ arc.

$$x1 = \left|\frac{x}{2\pi}\right|$$

Then compute sine as below

$$\sin(x) = \begin{cases} -sersin(2\pi - x1), if\ x1 \geq \frac{3.\pi}{2} \\ sersin(x1 - \pi), if\ x1 \geq \pi \\ sersin(\pi - x1), if\ x1 \geq \frac{\pi}{2} \end{cases}$$

Implemented Functions

Name	**_DFSIN**
Function	Perform Float Double Sine
Input values	**Float Acc**
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,R0..R15,AccH**
Time	----
Observations	----
Example	rcall _DFSIN call _DFSIN (chips >=16k)

BCD TO STRING (DFBCDTOS.INC File)

Description

Convert _DF_BCD SRAM Variable to string scientific formatted.

Implemented Functions

Name	**_DFBCDTOS**
Function	Convert BCD to String scientific formatted
Input values	**_DF_BCD SRAM variable**
Output values	**_DF_STR SRAM variable with string scientific formatted with 0 terminator**
Destroy	**Flags,R0..R15,Acc,AccH**
Time	----
Observations	----
Example	rcall _DFBCDTOS call _DFBCDTOS (chips >=16k)

FLOAT TO BCD (DFFTOBCD.INC File)

Description

Convert float double value in BCD format store in **_DF_BCD** SRAM variable.

Implemented Functions

Name	**_DFFTOBCD**
Function	Convert Float value to BCD
Input values	**Float Acc**
Output values	**_DF_BCD SRAM variable**
Destroy	**Flags,R0..R15,Acc,AccH**
Time	----
Observations	After call this routine you can use **_DFBCDTOS** to obtain result in string scientific format
Example	rcall _DFFTOBCD call _DFFTOBCD (chips >=16k)

FLOAT TO LONG (DFFTOL.INC File)

Description

Convert float double value to signed long value 32bits.

Implemented Functions

Name	**_DFFTOL**
Function	Convert Float value to Long
Input values	**Float Acc**
Output values	**AccTH:AccT:AccH:Acc 32bits output value** **Cy=1 if overflow occur**
Destroy	**Flags,Float OP1**
Time	----
Observations	----
Example	rcall _DFFTOL call _DFFTOL (chips >=16k)

HEADER MACROS (DOUBLE_FLOAT_MACROS.INC File)

(DOUBLE_FLOAT_HEADER.INC File)

Description

DOUBLE_FLOAT_HEADER is used to load all float point function and DOUBLE_FLOAT_HEADER to load all macros to used stacked float point function.

Implemented Functions

Stack Macros Names	Description
_FINIT	Initialize float point engine
_FDECST	Decrement Float Stack Point
_FINCST	Increment Float Stack Point
_FLD0	Float load 0 into Stack
_FLD1	Float load 1 into Stack
_FLD2	Float load 2 into Stack
_FLDPI2	Float load PI/2 into Stack
_FLDPI4	Float load PI/4 into Stack
_FLDPI	Float load PI into Stack
_FLD3PI2	Float load 3*PI/2 into Stack
_FLD2PI	Float load 2*PI into Stack
_FLDE	Float load e into Stack
_FLDSQRT2	Float load sqrt(2) into stack
_FLDSQRT2M1	Float load sqrt(2)-1 into stack
_FLDSQRT2P1	Float load sqrt(2)+1 into stack
_FLDII X	Float load X integer immediate into stack
_FLDI X	Float load X long integer immediate into stack
_FLDS S	Float load string S in Program Flash into stack
_FLDSS S	Float load string S in SRAM into stack
_FLDB M	Float load M Byte in SRAM into stack
_FLDI M	Float load M Integer in SRAM into stack
_FLDL M	Float load M Long in SRAM into stack
_FLDF M	Float load M single in SRAM into stack
_FLDD M	Float load M double in SRAM into stack
_FLD M	Same as _FLDD
_FSTB M	Float store byte into SRAM M
_FSTI M	Float store integer into SRAM M
_FSTL M	Float store long into SRAM M
_FSTF M	Float store single into SRAM M
_FSTD M	Float store double into SRAM M
_FST M	Same as _FSTD
_FSTBCD M	Float store BCD into SRAM M
_FSTBP M	Float store Byte into SRAM M and pop stack

_FSTIP M	Float store Integer into SRAM M and pop stack
_FSTLP M	Float store Long into SRAM M and pop stack
_FSTFP M	Float store single into SRAM M and pop stack
_FSTDP M	Float store double into SRAM M and pop stack
_FSTP M	Same as _FSTDP
_FSTBCDP M	Float store BCD into SRAM M and pop stack
_FADD	Float addition stack(0)+stack(1) and pop stack
_FSUB	Float addition stack(0)-stack(1) and pop stack
_FMUL	Float addition stack(0)*stack(1) and pop stack
_FDIV	Float addition stack(0)/stack(1) and pop stack
_FDIVFRAC	Float addition stack(0)/stack(1) return fraction part and pop stack
_FINV	Float point reciprocal 1/stack(0)
_FABS	Float point absolute of stack(0)
_FCHS	Float point multiply by -1 stack(0)
_FSQR	Float point square of stack(0)
_FSQRT	Float point square root of stack(0)
_FINT	Float point integer of stack(0),round to infinity
_FFIX	Float point integer of stack(0),round o zero
_FSERSIN	Float point partial sine of stack(0)
_FSERATN	Float point partial arctan of stack(0)
_FSIN	Float point sine of stack(0)
_FCOS	Float point cosine of stack(0)
_FTAN	Float point tangent of stack(0)
_FASIN	Float point arcsine of stack(0)
_FACOS	Float point arccosine of stack(0)
_FCOMP	Float Point compare stack(0) with stack(1)
_FBRANCH_EQ	Fload Point branch if equal
_FBRANCH_NEQ	Fload Point branch if not equal
_FBRANCH_GT	Fload Point branch if Great than
_FBRANCH_LT	Fload Point branch if less than
_FBRANCH_GTEQ	Fload Point branch if great than of equal
_FBRANCH_LTEQ	Fload Point branch if less than or equal

INFINIT RESULT (DFINF.INC File)

Description

Set infinities results.

Implemented Functions

Name	_DFUNDER
Function	Load float **Acc** with 0 and set underflow code
Input values	**None**
Output values	**Acc** underflow code **Float Acc** 0
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFUNDER call _DFUNDER (chips >=16k)

Name	_DFOVER
Function	Load float **Acc** with +-1.701412e+38 and set overflow code
Input values	**None**
Output values	**Acc** overflow code **Float Acc** +-1.701412e+38
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFOVER call _DFOVER (chips >=16k)

Name	_DFDIV0
Function	Load float **Acc** with max positive or negative +-1.701412e+38 and set overflow code
Input values	**Float Acc**
Output values	**Acc** overflow code **Float Acc** +-1.701412e+38
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFDIV0 call _DFDIV0 (chips >=16k)

INT (INT.INC File)

Description

Get integer part of float double value. Below examples of returned value for INT and FIX functions.

```
INT(-8.4)=-9
INT(-7)=-7
INT(5.45)=5
INT(9.9)=9
FIX(-6.5)=-6
FIX(6.5)=6
```

Implemented Functions

Name	**_DFINT**
Function	Float point get integer part
Input values	**Float Acc**
Output values	**Float Acc** **Acc exception code**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFINT call _DFINT (chips >=16k)

Name	**_DFFIX**
Function	Float point get integer part
Input values	**Float Acc**
Output values	**Float Acc** **Acc exception code**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFFIX call _DFFIX (chips >=16k)

LOAD STORE (DFLDXSTX.INC File)

Description

Functions to Load Constants and Load/Store Floats **Op1,Op1, Acc** into SRAM.

Implemented Functions

Name	**_DFLD0**
Function	Float point load zero into **Float Acc**
Input values	None
Output values	**Float Acc =0** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLD0 call _DFLD0 (chips >=16k)

Name	**_DFLD1**
Function	Float point load one(1) into **Float Acc**
Input values	None
Output values	**Float Acc =1** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLD1 call _DFLD1 (chips >=16k)

Name	_DFLD2
Function	Float point load two(2) into **Float Acc**
Input values	None
Output values	**Float Acc =2** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLD2 call _DFLD2 (chips >=16k)

Name	_DFLD10
Function	Float point load 10 into **Float Acc**
Input values	None
Output values	**Float Acc =10** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLD10 call _DFLD10 (chips >=16k)

Name	_DFLDE
Function	Float point load e=2,71828182845904524 into **Float Acc**
Input values	None
Output values	**Float Acc =2,71828182845904524** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLDE call _DFLDE (chips >=16k)

Name	_DFLDSQRT2
Function	Float point load sqrt(2)=1,414213562373 into **Float Acc**
Input values	None
Output values	**Float Acc =1,414213562373** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLDSQRT2 call _DFLDSQRT2 (chips >=16k)

Name	_DFLDSQRT2M1
Function	Float point load sqrt(2)-1=0,414213562373 into **Float Acc**
Input values	None
Output values	**Float Acc =0,414213562373** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLDSQRT2M1 call _DFLDSQRT2M1 (chips >=16k)

Name	_DFLDSQRT2P1
Function	Float point load sqrt(2)+1=2,414213562373 into **Float Acc**
Input values	None
Output values	**Float Acc =2,414213562373** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLDSQRT2P1 call _DFLDSQRT2P1 (chips >=16k)

Name	_DFLDPI2
Function	Float point load pi/2=1,5707963267948966 into **Float Acc**
Input values	None
Output values	**Float Acc =1,5707963267948966** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLDPI2 call _DFLDPI2 (chips >=16k)

Name	_DFLDPI4
Function	Float point load pi/4=0,7853981633974483 into **Float Acc**
Input values	None
Output values	**Float Acc = 0,7853981633974483** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLDPI4 call _DFLDPI4 (chips >=16k)

Name	_DFLDPI
Function	Float point load pi=3,14159265358979324 into **Float Acc**
Input values	None
Output values	**Float Acc =3,14159265358979324** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLDPI call _DFLDPI (chips >=16k)

Name	_DFLD3PI2
Function	Float point load 3*pi/2=4,71238898038468985 into **Float Acc**
Input values	None
Output values	**Float Acc =4,71238898038468985** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLD3PI2 call _DFLD3PI2 (chips >=16k)

Name	_DFLD2PI
Function	Float point load 2pi=6,28318530717958648 into **Float Acc**
Input values	None
Output values	**Float Acc =6,28318530717958648** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLD2PI call _DFLD2PI (chips >=16k)

Name	_DFLDZ
Function	Load Float Acc pointed by Z register
Input values	Z → Float Point in SRAM
Output values	**Float Acc**
Destroy	**Flags**
Time	----
Observations	----
Example	Load **Float Acc** with value_sram Ldiw Z,value_sram rcall _DFLDZ call _DFLDZ (chips >=16k)

Name	_DFLDDC
Function	Load **Float Acc** pointed by Z register into Flash
Input values	Z → Float Point in FLASH
Output values	**Float Acc**
Destroy	**Flags**
Time	----
Observations	----
Example	Load **Float Acc** with value_flash Ldiw Z,value_flash*2 rcall _DFLDDC call _DFLDDC (chips >=16k)

Name	_DFSTZ
Function	Store **Float Acc** pointed by Z register
Input values	Z → Float Point in SRAM **Float Acc** value
Output values	**---**
Destroy	**Flags,Z**
Time	----
Observations	----
Example	Store **Float Acc** into value_sram Ldiw Z,value_sram rcall _DFSTZ call _DFSTZ (chips >=16k)

Name	_DFLDOP1Z
Function	Load **Float Op1** pointed by Z register
Input values	Z → Float Point in SRAM
Output values	**Float Op1**
Destroy	**Flags,Z**
Time	----
Observations	----
Example	Load **Float Op1** from value_sram Ldiw Z,value_sram rcall _DFLDOP1Z call _DFLDOP1Z (chips >=16k)

Name	_DFLDOP2Z
Function	Load **Float Op2** pointed by Z register
Input values	Z → Float Point in SRAM
Output values	**Float Op2**
Destroy	**Flags,Z**
Time	----
Observations	----
Example	Load **Float Op2** from value_sram Ldiw Z,value_sram rcall _DFLDOP2Z call _DFLDOP2Z (chips >=16k)

Name	_DFACCOP1
Function	Copy **Float Acc** to **Float Op1**
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	Move **Float Acc** To **Float Op1** rcall _DFACCOP1 call _DFACCOP1 (chips >=16k)

Name	_DFOP1ACC
Function	Copy **Float Op1** to **Float Acc**
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	Move **Float Op1** To **Float Acc** rcall _DFOP1ACC call _DFOP1ACC (chips >=16k)

Name	_DFACCOP2
Function	Copy **Float Acc** to **Float Op2**
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	Move **Float Acc** To **Float Op2** rcall _DFACCOP2 call _DFACCOP2 (chips >=16k)

Name	_DFOP2ACC
Function	Copy **Float Op2** to **Float Acc**
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	Move **Float Op2** To **Float Acc** rcall _DFOP2ACC call _DFOP2ACC (chips >=16k)

MACRO	_DFPUSHACC
Function	Push **Float Acc** into Stack pointer
Example	_DFPUSHACC

MACRO	_DFPOPACC
Function	Pop **Float Acc** From Stack pointer
Example	_DFPopACC

MACRO	_DFPUSHOP1
Function	Push **Float Op1** into Stack pointer
Example	_DFPUSHOP1

MACRO	_DFPOPOP1
Function	Pop **Float Op1** From Stack pointer
Example	_DFPOPOP1

MACRO	_DFPUSHOP2

Function	Push **Float Op2** into Stack pointer
Example	_DFPUSHOP2

MACRO	**_DFPOPOP2**
Function	Pop **Float Op2** From Stack pointer
Example	_DFPOPOP2

LONG TO FLOAT (DFLTOF.INC File)

Description

Convert integer quantities byte, word or long to double float point.

Implemented Functions

Name	**_DFLTOF**
Function	Convert long signed 32bits value to double float point
Input values	**AccTH:AccT:AccH:Acc** long value 32 bits
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLTOF call _DFLTOF (chips >=16k)

Name	**_DFITOF**
Function	Convert Integer signed 16bits value to double float point
Input values	**AccH:Acc** integer value 16 bits
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFITOF call _DFITOF (chips >=16k)

Name	**_DFATOF OR _DFBTOF**
Function	Convert Byte signed 8bits value to double float point
Input values	**Acc** 8bits value
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFLTOF call _DFLTOF (chips >=16k)

SERIES

ATN – ArcTangent series (SERIE_ATN.INC File)

Description

Compute a partial ArcTangent function using Taylor serie describe below. The range of value of this function is $[-\sqrt{2}+1, \sqrt{2}+1]$, values out this range lost precision.

A original Taylor series for ArcTangent follow:

$$\operatorname{atan}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} \dots$$

But because speed reason the equation is rewritten as following

$$sq = x^2$$

$$v1 = \left(\left(\left((p4.sq + p3).sq + p2\right).sq + p1\right).sq + p0\right)$$

$$v2 = \frac{v1}{\left(\left(\left((sq + q4).sq + q3\right).sq + q2\right).sq + q1\right).sq + q0)}$$

$$\operatorname{atan}(x) = v2.x$$

$numerador\ coefficients$

$p4\ =\ \ 16{,}1536412982230228262$
$p3\ =\ \ 268{,}42548195503973794141$
$p2\ =\ 1153{,}0293515404850115428136$
$p1\ =\ 1780{,}406316433196971054645 87$
$p0\ =\ \ 896{,}78597403663861959987488$
$quotient\ coefficients$

$q4 = \ \ 58,956970508444622222791$
$q3 = \ \ 536,265374031215315104235$
$q2 = 1666,783814881633718452179 8$
$q1 = 2079,334974445409812872759 26$
$q0 = \ \ 896,785974036638619624811 62$

$coefficients\ are\ \#5077\ from\ Hart\ \&\ Cheney.(19.56D)$

Implemented Functions

Name	***_DFSERATN***
Function	Compute Partial ArcTangent
Input values	**Float Acc**
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,r0..r15,AccH**
Time	----
Observations	----
Example	rcall _DFSERATN call _DFSERATN (chips >=16k)

LN – Natural Logarithm (SERIE_LN.INC File)

Description

Compute a partial natural logarithm function using Maclaurin series describe below. The range of value of this function is [1,2], values out this range lost precision.

Modified Maclaurin series for more fast conversion of results of natural logarithm follow:

$$\ln\left(\frac{x-1}{x+1}\right) = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \frac{x^9}{9} + \frac{x^{11}}{11} \ldots\right)$$

But to obtain more fast algorithm the equation is rewritten as following

$$ln\left(\frac{x-1}{x+1}\right) = 2.x(1 + x^2(\frac{1}{3} + x^2(\frac{1}{5} + x^2(\frac{1}{7} + x^2(\frac{1}{9} + x^2(\ldots$$

Implemented Functions

Name	**_DFSERATN**
Function	Compute Partial Natural Logarithm
Input values	**Float Acc**
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,r0..r15,AccH**
Time	----
Observations	----
Example	rcall _DFSERLN call _DFSERLN (chips >=16k)

SERIE SINE (SERIE_SINE.INC File)

Description

Compute a partial Sine function using Maclaurin series describe below. The range of value of this function is $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$, values out this range lost precision.

A original Maclaurin series for sine follow:

$$\sin(\mathrm{x}) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} \ldots$$

But to obtain more fast algorithm equation is rewritten as following

$$\sin(x) = x(1 + x^2(\frac{1}{3!} + x^2(\frac{1}{5!} + x^2(\frac{1}{7!} + x^2(\frac{1}{9!} + x^2(\ldots$$

Implemented Functions

Name	**_DFSERATN**
Function	Compute Partial Sine
Input values	**Float Acc**
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,r0..r15,AccH**
Time	----
Observations	----
Example	rcall _DFSERSIN call _DFSERSIN (chips >=16k)

SQUARE ROOT (DFSQRT.INC File)

Description

Compute a square root function using Newton/Raphson approximation algorithm.

$$Let\ x_0 = x\ is\ my\ first\ aproximation, then$$
$$x_{n+1} = \frac{\left(\frac{x}{x_n} + x_n\right)}{2}$$

Repeat above operation until $x_{x+1} = x_n$, in that instant x_{n+1} is equal to square root of x.

Others considerations, the value x stored in memory use a base 2 using the following format.

$x = m_x.2^{E_x}$ where m_x is the mantissa of x and E_x is the exponent of x base 2. This well known, the below simplification occurs.

$$\sqrt{x} \rightarrow \sqrt{m_x.2^{E_x}} \rightarrow \sqrt{m_x}.\sqrt{2^{E_x}}$$

When E_x is even then

$$\sqrt{x} = \sqrt{m_x}.2^{\frac{E_x}{2}}$$

When E_x is odd then

$$\sqrt{x} = \sqrt{m_x}.2^{\frac{E_{x-1}}{2}}.\sqrt{2}$$

How the base is 2 de maximum value for de m_x is 2, the number of interaction of Newton/Raphson approximation is 6 for a 16 decimal digits number found experimentally.

Implemented Functions

Name	**_DFSQRT**
Function	Compute Square root
Input values	**Float Acc**
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,r0..r15,AccH**
Time	----
Observations	----
Example	rcall _DFSQRT call _DFSQRT (chips >=16k)

STRING TO FLOAT (DFSTOF.INC File)

Description

Convert a String number in ASCII format store in SRAM or FLASH to float double value. The string must be zero ended \0.

Implemented Functions

Name	**_DFSSTOF**
Function	Convert a string number in SRAM to float double value
Input values	**Z ->** String into SRAM
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,r0..r15,AccH**
Time	----
Observations	----
Example	Convert sram_string_value to float point Ldiw Z,sram_string_value rcall _DFSSTOF call _DFSSTOF (chips >=16k)

Name	**_DFSTOF**
Function	Convert a string number in FLASH to float double value
Input values	**Z ->** String into SRAM
Output values	**Float Acc** **Acc** exception code
Destroy	**Flags,r0..r15,AccH**
Time	----
Observations	----
Example	Convert flash_string_value to float point Ldiw Z,flash_string_value*2 rcall _DFSTOF call _DFSTOF (chips >=16k)

TAN - Tangent (TAN.INC File)

Description

Compute a Tangent of float double value. Using the follow equation:

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Implemented Functions

Name	**_DFSSTOF**
Function	*Compute Tangent*
Input values	***Float Acc*** *input value*
Output values	***Float Acc*** ***Acc*** *exception code*
Destroy	***Flags,r0..r15,AccH***
Time	----
Observations	----
Example	*rcall _DFTAN* *call _DFTAN (chips >=16k)*

UNARY (UNARY.INC File)

Description

Perform float double function that use one operator.

Implemented Functions

Name	**_DFABS**
Function	Float double absolute
Input values	**Float Acc** input value
Output values	**Float Acc =0** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFABS call _DFABS (chips >=16k)

Name	**_DFCHS**
Function	Float double change sign
Input values	**Float Acc** input value
Output values	**Float Acc =0** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFCHS call _DFCHS (chips >=16k)

Name	**_DFINV**
Function	Float double reciprocal value 1/x
Input values	**Float Acc** input value
Output values	**Float Acc =0** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFINV call _DFINV (chips >=16k)

Name	_DFSQR
Function	Float double square
Input values	**Float Acc** input value
Output values	**Float Acc =0** **Acc FPOK**
Destroy	**Flags**
Time	----
Observations	----
Example	rcall _DFSQR call _DFSQR (chips >=16k)

INTEGER

ATOS (ATOS.INC File)

Description

Convert signed byte to string ASCII with fixed size 4 chars and with left zero removal with 0 terminator \0.

Implemented Functions

Name	***_ATOS or _BTOS***
Function	Convert signed byte to ASCII string
Input values	**Acc** signed input value
Output values	**Z→** output string \0
Destroy	Flags
Time	----
Observations	
Example	Convert Acc=45 to String Ldi **Acc**,45 Rcall _ATOS Call _ATOS

CHECKSUM16 (CHECKSUM16.INC File)

Description

Compute 16 string of data pointed by z register. May be used to compute IP checksum.

Implemented Functions

Name	***_CHECKSUM16***
Function	Convert signed byte to ASCII string
Input values	**AccH:Acc** number of words of string **Z->** data in SRAM
Output values	**AccH:Acc checksum**
Destroy	Flags
Time	----
Observations	if user use this routine only for check purpose the return value must be 0xffff but if used to compute new checksum field the returned value must be complemented using 1 complement
Example	Compute checksum of string_SRAM of 100 words Ldiw Z,string_SRAM Ldiaw 100 Rcall _CHECKSUM16 Call _CHECKSUM16 (chips>=16k)

DIV_S32S32S32 (DIV_S32S32S32.INC File)

Description

Divide signed 32bits by signed 32bits and remainder quotient of 32bits and rest 32bits.

Implemented Functions

Name	***_DIV_S32S32S32***
Function	Divide signed 32bits by 32bits and remainder quotient of 32bits and rest of 32bits
Input values	**R3:R2:R1:R0** Dividend **R7:R6:R5:R4** Divisor
Output values	**R3:R2:R1:R0** Quotient **R8:R9:R10:R11** Rest
Destroy	Flags
Time	Min=566 Max=662 clocks
Observations	---

Example	Divide 100000 by 456
	
	Ldial 100000
	Mov r0,**Acc**
	Mov r1,**AccH**
	Mov r2,**AccT**
	Mov r3,**AccTH**
	Ldial 456
	Mov r4,**Acc**
	Mov r5,**AccH**
	Mov r6,**AccT**
	Mov r7,**AccTH**
	
	rcall _DIV_S32S32S32
	Call _DIV_S32S32S32 (chips>=16k)

DIV_U8U8U8 (DIV_U8U8U8.INC File)

Description

Divide unsigned 8bits by unsigned 8bits result unsigned 8bits.

Implemented Functions

Name	***_DIV_U8U8U8U8***
Function	Divide unsigned 8bits by unsigned 8bits result unsigned 8bits
Input values	**R0** Dividend **R1** Divisor
Output values	**R0** Quotient **R2** Rest
Destroy	Flags
Time	Min=77 Max=77 clocks
Observations	---

Example	Divide 100 by 7
	
	Ldi **Acc**,100
	Mov r0,**Acc**
	Ldi **Acc,7**
	Mov **r1,Acc**
	
	rcall _DIV_U8U8U8
	Call _DIV_U8U8U8 (chips>=16k)

DIV_U16U16U16 (DIV_U16U16U16.INC File)

Description

Divide unsigned 16bits by unsigned 16bits result unsigned 16bits.

Implemented Functions

Name	***_DIV_U16U16U16***
Function	Divide unsigned 16bits by unsigned 16bits result unsigned 16bits
Input values	**R1:R0** Dividend **R3:R2** Divisor
Output values	**R1:R0** Quotient **R5:R4** Rest
Destroy	Flags
Time	Min=192 Max=208 clocks
Observations	---

Example	Divide 1000 by 37
	
	Ldiaw 1000
	Mov r0,**Acc**
	Mov r1,**AccH**
	Ldiaw **37**
	Mov r2,**Acc**
	Mov r3,**AccH**
	
	rcall _DIV_U16U16U16
	Call _DIV_U16U16U16 (chips>=16k)

DIV_U16U16U16F16 (DIV_U16U16U16F16.INC File)

Description

Divide unsigned 16bits by unsigned 16 bits result unsigned 16bit integer part and 16bits fraction part.

Implemented Functions

Name	***_DIV_U16U16U16F16***
Function	Divide unsigned 16bits by unsigned 16bits result unsigned 16bits and fractional part 16bits
Input values	**R1:R0** Dividend **R3:R2** Divisor
Output values	**R1:R0** Quotient fraction part **R3:R2** Quotient integer part
Destroy	Flags
Time	Min=451 Max=484 clocks
Observations	---

Example	Divide 1000 by 37
	
	Ldiaw 1000
	Mov r0,**Acc**
	Mov r1,**AccH**
	Ldiaw **37**
	Mov r2,**Acc**
	Mov r3,**AccH**
	
	rcall _DIV_U16U16U16F16
	Call _DIV_U16U16U16F16 (chips>=16k)

DIV_U32U32U32 (DIV_U32U32U32.INC File)

Description

Divide unsigned 32bits by unsigned 32 bits result unsigned 32bits.

Implemented Functions

Name	*_DIV_U32U32U32*
Function	Divide unsigned 32bits by unsigned 32bits result unsigned 32bits
Input values	**R3:R2:R1:R0** Dividend **R7:R6:R5:R4** Divisor
Output values	**R3:R2:R1:R0** Quotient **R11:R10:R9:R8 Rest**
Destroy	Flags
Time	Min=566 Max=662 clocks
Observations	---

Example	Divide 100000 by 456
	
	Ldial 100000
	Mov r0,**Acc**
	Mov r1,**AccH**
	Mov r2,**AccT**
	Mov r3,**AccTH**
	Ldial 456
	Mov r4,**Acc**
	Mov r5,**AccH**
	Mov r6,**AccT**
	Mov r7,**AccTH**
	
	rcall _DIV_U32U32U32
	Call _DIV_U32U32U32 (chips>=16k)

DUMPHEX (DUMPHEX.INC File)

Description

Convert byte to ASCII two(2) digits hexadecimal value.

Implemented Functions

Name	***_DUMPHEX***
Function	Convert byte to STRING ASCII hexadecimal
Input values	**Acc** input value **Z->** SRAM area to receive hex ASCII value
Output values	**Z->** SRAM after 2 bytes (2 hex digits)
Destroy	Flags
Time	---
Observations	---
Example	Convert 100 to hex Ldi Acc,100 rcall _DUMPHEX Call _DUMPHEX (chips>=16k) Result "64" in SRAM

Name	***_TOHEX2***
Function	Same as _DUMPHEX but zero ended \0
Input values	**Acc** input value
Output values	**Z->** SRAM area that receive hex ascii digits
Destroy	Flags
Time	---
Observations	---
Example	Convert 100 to hex Ldi **Acc**,100 rcall _TOHEX2 Call _TOHEX2 (chips>=16k) Result "64" in SRAM pointed by **Z**

INT_ATN (INT_ATN.INC File)

Description

Compute integer ArcTangent(X/Y).

Implemented Functions

Name	***_INT_ATN_XY***
Function	Compute integer ArcTangent(X/Y).
Input values	**X 1st** value **Y 2nd** value
Output values	**AccH:Acc** angle range -90..90 degrees
Destroy	Flags
Time	---
Observations	---

Example	Get Integer arctangent between 100 and 45
	
	Ldiw X,100
	Ldiw Y,45
	rcall _INT_ATN_XY
	Call _INT_ATN_XY (chips>=16k)

INT_ATN2PTS (INT_ATN2PTS.INC File)

Description

Compute integer ArcTangent between two points p1(x,y) and p2(X/Y).

Implemented Functions

Name	***_INT_ATN2PTS***
Function	Compute integer arctangent between two points p1(x,y) and p2(x,y)
Input values	**AccH:Acc, AccTH:AccT** p1(x,y) **X,Y** p2(x,y)
Output values	**AccH:Acc** angle range 0..359 in degrees
Destroy	Flags
Time	---
Observations	---

Example	Get Integer arctangent between p1(100,50) and p2(200,70)
	
	Ldiaw 100
	Ldiawt 50
	Ldiw X,200
	Ldiw Y,70
	rcall _INT_ATN2PTS
	Call _INT_ATN2PTS (chips>=16k)

IPTOS (IPTOS.INC File)

Description

Convert 32bits number to string ASCII formatted as aaa.bbb.ccc.ddd \0

Implemented Functions

Name	***_IPTOS***
Function	Convert 32bits number to string IP formatted.
Input values	**X,Y** IP number
Output values	**Z ->** SRAM STRING ASCII FORMATTED IP
Destroy	Flags
Time	---
Observations	---

Example	Convert to string a -1062680549 32bits value to formatted IP number.
	
	Ldiw X,lwrd(-1062680549)
	Ldiw Y,hwrd(-1062680549)
	rcall _IPTOS
	Call _IPTOS (chips>=16k)
	
	Result string "192.168.200.27"\0

ITOS (ITOS.INC File)

Description

Convert 16bits signed value to string ASCII with 0 ended \0.

Implemented Functions

Name	***_ITOS***
Function	Convert 16bits unsigned value to string ASCII
Input values	**AccH:Acc** Signed value
Output values	**Z ->** SRAM STRING ASCII +\0
Destroy	Flags
Time	---
Observations	---
Example	Convert 1234 to string. Ldiaw 1234 rcall _ITOS Call _ITOS (chips>=16k) Result string “1234”\0

IXTOS (IXTOS.INC File)

Description

Convert byte,word,integer,long to string +\0.

Implemented Functions

Name	***_I_USE_SIZE***
Function	Set size for formatted output string
Input values	**Acc** Size
Output values	None
Destroy	Flags
Time	---
Observations	---
Example	Set output string size to 4. Ldi Acc,4 rcall _I_USE_SIZE Call _I_USE_SIZE (chips>=16k)

Name	***_I_USE_PLUS***
Function	Set '+' sign before positive number
Input values	**Acc** =1 if use '+' before positive number 0 otherwise
Output values	None
Destroy	Flags
Time	---
Observations	---
Example	Set not plus signal in before number Ldi Acc,0 rcall _I_USE_PLUS Call _I_USE_PLUS (chips>=16k)

Name	***_I_USE_THOUSAND***
Function	Set Thousand separator of formatted numbers
Input values	**Acc** =1 used separator 0 otherwise
Output values	None
Destroy	Flags
Time	---
Observations	---
Example	Set use of thousand separator. Ldi **Acc**,1 rcall _I_USE_THOUSAND Call _I_USE_THOUSAND (chips>=16k)

Name	***_I_USE_SEP_CHAR***
Function	Set character for thousand separator
Input values	**Acc** =1 for point =0 for comma
Output values	None
Destroy	Flags
Time	---
Observations	---
Example	Set thousand separator character as comma. Ldi **Acc**,0 rcall _I_USE_SEP_CHAR Call _I_USE_SEP_CHAR (chips>=16k)

Name	***_IBTOS***
Function	Integer signed byte to string automatic length
Input values	**Acc** Signed byte
Output values	Z-> formatted automatic length
Destroy	Flags
Time	---
Observations	---
Example	Convert 45 to string automatic length Ldi **Acc**,15 rcall _IBTOS Call _IBTOS (chips>=16k) Result “45\0”

Name	_IBTOSF
Function	Integer signed byte to string formatted length
Input values	**Acc** Signed byte
Output values	Z-> formatted length string
Destroy	Flags
Time	---
Observations	---

Example	Convert 45 to string formatted length=5
	
	Ldi **Acc**,5
	Call _I_USE_SIZE
	Ldi **Acc**,45
	rcall _IBTOS
	Call _IBTOS (chips>=16k)
	
	Result " 45\0"

Name	_IUTOS
Function	Integer unsigned word to string automatic length
Input values	**AccH:Acc** Signed word
Output values	Z-> formatted automatic string
Destroy	Flags
Time	---
Observations	---
Example	Convert 1245 to string Ldiaw 1245 rcall _IUTOS Call _IUTOS (chips>=16k) Result "1245\0"

Name	_ITOS
Function	Integer signed word to string automatic length
Input values	**AccH:Acc** Signed word
Output values	Z-> formatted automatic string
Destroy	Flags
Time	---
Observations	---
Example	Convert -1245 to string Ldiw -1245 rcall _ITOS Call _ITOS (chips>=16k) Result "-1245\0"

Name	_ITOSF
Function	Integer signed word to string formatted length
Input values	**AccH:Acc** Signed word
Output values	**Z->** formatted string
Destroy	Flags
Time	---
Observations	---

Example	Convert 1245 to string formatted size = 8, use thousand separator and plus sign
	
	Ldi **Acc**,8
	Call _I_USE_SIZE
	Ldi **Acc**,1
	Call _I_USE_THOUSAND
	Ldi **Acc**,1
	Call _I_USE_SEP_CHAR
	Ldiaw 1245
	rcall _ITOSF
	Call _ITOSF (chips>=16k)
	
	Result " +1,245\0"

Name	_IUTOSF
Function	Integer unsigned word to string formatted length
Input values	**AccH:Acc** unsigned word
Output values	**Z->** formatted string
Destroy	Flags
Time	---
Observations	---

Example	Convert 1245 to string formatted size = 8, use thousand separator and plus sign
	
	Ldi **Acc**,8
	Call _I_USE_SIZE
	Ldi **Acc**,1
	Call _I_USE_THOUSAND
	Ldi **Acc**,1
	Call _I_USE_SEP_CHAR
	Ldiaw 1245
	rcall _IUTOSF
	Call _IUTOSF (chips>=16k)
	
	Result " +1,245\0"

Name	_IULTOS
Function	Long Integer unsigned word to string automatic length
Input values	**AccTH:AccT:AccH:Acc** unsigned long
Output values	**Z->** formatted string
Destroy	Flags
Time	---
Observations	---

Example	Convert 12345678 to string automatic size = 12, use thousand separator and plus sign
	
	Ldi **Acc**,12
	Call _I_USE_SIZE
	Ldi **Acc**,1
	Call _I_USE_THOUSAND
	Ldi **Acc**,1
	Call _I_USE_SEP_CHAR
	Ldial 12345678
	rcall _IULTOS
	Call _IULTOS (chips>=16k)
	
	Result "+12,345,678\0"

Name	***_ILTOS***
Function	Long Integer signed word to string automatic length
Input values	**AccTH:AccT:AccH:Acc** signed long
Output values	**Z->** formatted string
Destroy	Flags
Time	---
Observations	---

Example	Convert -12345678 to string automatic size = 12, use thousand separator and plus sign
	
	Ldi **Acc**,12
	Call _I_USE_SIZE
	Ldi **Acc**,1
	Call _I_USE_THOUSAND
	Ldi **Acc**,1
	Call _I_USE_SEP_CHAR
	Ldial -12345678
	rcall _IULTOS
	Call _IULTOS (chips>=16k)
	
	Result "-12,345,678\0"

Name	***_ILTOSF***
Function	Long Integer signed word to string formatted length
Input values	**AccTH:AccT:AccH:Acc** signed long
Output values	**Z->** formatted string
Destroy	Flags
Time	---
Observations	---

Example	Convert -12345678 to string formatted size = 12, use thousand separator and plus sign
	
	Ldi **Acc**,12
	Call _I_USE_SIZE
	Ldi **Acc**,1
	Call _I_USE_THOUSAND
	Ldi **Acc**,1
	Call _I_USE_SEP_CHAR
	Ldial -12345678
	rcall _IULTOS
	Call _IULTOS (chips>=16k)
	
	Result " -12,345,678\0"

Name	_IULTOSF
Function	Long Integer signed word to string formatted length
Input values	AccTH:AccT:AccH:Acc signed long
Output values	Z-> formatted string
Destroy	Flags
Time	---
Observations	---

Example	Convert 12345678 to string formatted size = 12, use thousand separator and plus sign
	
	Ldi Acc,12
	Call _I_USE_SIZE
	Ldi Acc,1
	Call _I_USE_THOUSAND
	Ldi Acc,1
	Call _I_USE_SEP_CHAR
	Ldial 12345678
	rcall _IULTOS
	Call _IULTOS (chips>=16k)
	
	Result " +12,345,678\0"

MUL_S8S8S8 (MUL_S8S8S8.INC File)

Description

Multiply signed 8bits by signed 8bits result signed 8 bits

Implemented Functions

Name	***_MUL_S8S8S8***
Function	Multiply signed 8bits by signed 8bits result signed 8bits
Input values	**R0** 1st operand **R1** 2nd operand
Output values	**R2** Product
Destroy	Flags,R1,R3
Time	Min=43 clocks
Observations	---

Example	Multiply 5 by 7
	
	Ldi **Acc**,5
	Mov r0,**Acc**
	Ldi **Acc**,7
	Mov r1,**Acc**
	rcall _MUL_S8S8S8
	Call _MUL_S8S8S8 (chips>=16k)

MUL_S8S8S16 (MUL_S8S8S16.INC File)

Description

Multiply signed 8bits by signed 8bits result signed 16 bits

Implemented Functions

Name	***_MUL_S8S8S16***
Function	Multiply signed 8bits by signed 8bits result signed 16bits
Input values	**R0** 1st operand **R1** 2nd operand
Output values	**R3:R2** Product
Destroy	Flags,R1,R4,R5
Time	Min=79 clocks
Observations	---

Example	Multiply 100 by 200
	
	Ldi **Acc**,100
	Mov r0,**Acc**
	Ldi **Acc**,200
	Mov r1,**Acc**
	rcall _MUL_S8S8S16
	Call _MUL_S8S8S16 (chips>=16k)

MUL_S16S16S16 (MUL_S16S16S16.INC File)

Description

Multiply signed 16bits by signed 16bits result signed 16 bits

Implemented Functions

Name	***_MUL_S16S16S16***
Function	Multiply signed 16bits by signed 16bits result signed 16bits
Input values	**R1:R0** 1st operand **R3:R2** 2nd operand
Output values	**R5:R4** Product
Destroy	Flags,R0,R1,R2,R3,R6
Time	Min=180 Max=211 clocks
Observations	---

Example	Multiply 100 by 200
	
	Ldiaw 100
	Mov r0,**Acc**
	MOV r1,**AccH**
	Ldiaw 200
	Mov r2,**Acc**
	Mov r3,**AccH**
	rcall _MUL_S16S16S16
	Call _MUL_S16S16S16 (chips>=16k)

MUL_S16S16S32 (MUL_S16S16S32.INC File)

Description

Multiply signed 16bits by signed 16bits result signed 32 bits

Implemented Functions

Name	***_MUL_S16S16S32***
Function	Multiply signed 16bits by signed 16bits result signed 32bits
Input values	**R1:R0** 1st operand **R3:R2** 2nd operand
Output values	**R7:R6:R5:R4** Product
Destroy	Flags,R0,R1,R2,R3
Time	Max=284 clocks
Observations	---

Example	Multiply 1000 by 2000
	
	Ldiaw 1000
	Mov r0,**Acc**
	MOV r1,**AccH**
	Ldiaw 2000
	Mov r2,**Acc**
	Mov r3,**AccH**
	rcall _MUL_S16S16S32
	Call _MUL_S16S16S32 (chips>=16k)

MUL_S32S32S32 (MUL_S32S32S32.INC File)

Description

Multiply signed 32bits by signed 32bits result signed 32 bits

Implemented Functions

Name	***_MUL_S32S32S32***
Function	Multiply signed 32bits by signed 32bits result signed 32bits
Input values	**R3:R2:R1:R0** 1st operand **R7:R6:R5:R2** 2nd operand
Output values	**R11:R10:R9:R8** Product
Destroy	Flags,R0,R1,R2,R3,R4,R5,R6,R7,R12
Time	Min=416 Max=571 clocks
Observations	---

Example	Multiply 12345 by 45000
	
	Ldial 12345
	Mov r0,**Acc**
	Mov r1**,AccH**
	Mov r2,**AccT**
	Mov r3,**AccTH**
	Ldial 45000
	Mov r4,**Acc**
	Mov r5**,AccH**
	Mov r6**,AccT**
	Mov r7**,AccTH**
	rcall _MUL_S32S32S32
	Call _MUL_S32S32S32 (chips>=16k)

MUL_U8U8U8 (MUL_U8U8U8.INC File)

Description

Multiply unsigned 8bits by unsigned 8bits result unsigned 8 bits

Implemented Functions

Name	***_MUL_U8U8U8***
Function	Multiply unsigned 8bits by unsigned 8bits result unsigned 8bits
Input values	**R0** 1st operand **R1** 2nd operand
Output values	**R2** Product
Destroy	Flags,R1
Time	Max=28 clocks
Observations	---

Example	Multiply 5 by 7
	
	Ldi **Acc**,5
	Mov r0,**Acc**
	Ldi **Acc**,7
	Mov r1,**Acc**
	rcall _MUL_U8U8U8
	Call _MUL_U8U8U8 (chips>=16k)

MUL_U8U8U16 (MUL_U8U8U16.INC File)

Description

Multiply unsigned 8bits by unsigned 8bits result unsigned 16 bits

Implemented Functions

Name	***_MUL_U8U8U16***
Function	Multiply unsigned 8bits by unsigned 8bits result unsigned 16bits
Input values	**R0** 1st operand **R1** 2nd operand
Output values	**R3:R2** Product
Destroy	Flags,R1,R4
Time	Min=63 Max=87 clocks
Observations	---

Example	Multiply 100 by 200
	
	Ldi **Acc**,100
	Mov r0,**Acc**
	Ldi **Acc**,200
	Mov r1,**Acc**
	rcall _MUL_U8U8U16
	Call _MUL_U8U8U16 (chips>=16k)

MUL_U16U16U16 (MUL_U16U16U16.INC File)

Description

Multiply unsigned 16bits by unsigned 16bits result unsigned 16 bits

Implemented Functions

Name	***_MUL_S16S16S16***
Function	Multiply unsigned 16bits by unsigned 16bits result unsigned 16bits
Input values	**R1:R0** 1st operand **R3:R2** 2nd operand
Output values	**R5:R4** Product
Destroy	Flags,R0,R1,R2,R3
Time	Min=158 Max=174 clocks
Observations	---

Example	Multiply 100 by 200
	
	Ldiaw 100
	Mov r0,**Acc**
	MOV r1**,AccH**
	Ldiaw 200
	Mov r2,**Acc**
	Mov r3**,AccH**
	rcall _MUL_U16U16U16
	Call _MUL_U16U16U16 (chips>=16k)

MUL_U16U16X10 (MUL_U16U16X10.INC File)

Description

Multiply unsigned 16bits number by 10 result back unsigned 16 bits

Implemented Functions

Name	***_MUL_U16U16U32***
Function	Multiply unsigned 16bits number by 10 result back unsigned 16 bits
Input values	**R1:R0** 1st operand
Output values	**R1:R0** Result
Destroy	Flags,R2,R3
Time	Max=14 clocks
Observations	---

Example	Multiply 1000 by 10
	
	Ldiaw 1000
	Mov r0,**Acc**
	MOV r1**,AccH**
	rcall _MUL_U16U16X10
	Call _MUL_U16U16X10 (chips>=16k)

MUL_U32U32U32 (MUL_U32U32X32.INC File)

Description

Multiply unsigned 32bits by unsigned 32bits result unsigned 32 bits.

Implemented Functions

Name	***_MUL_U32U32U32***
Function	Multiply unsigned 32bits by unsigned 32 bits result unsigned 32 bits
Input values	**R3:R2:R1:R0** 1st operand **R7:R6:R5:R4** 2nd operand
Output values	**R11:R10:R9:R8** 2nd operand
Destroy	Flags,R0,R1,R2,R3,R4,R5,R6,R7
Time	Min=397 Max=525 clocks
Observations	---

Example	Multiply 1000 by 1000
	
	Ldial 1000
	Mov r0,**Acc**
	Mov r1**,AccH**
	Mov r2,**AccT**
	Mov r3,**AccTH**
	Mov r4,**Acc**
	Mov r5**,AccH**
	Mov r6,**AccT**
	Mov r7,**AccTH**
	rcall _MUL_U32U32U32
	Call _MUL_U32U32U32 (chips>=16k)

MUL_U32U32U64 (MUL_U32U32X64.INC File)

Description

Multiply unsigned 32bits by unsigned 32bits result unsigned 64 bits.

Implemented Functions

Name	***_MUL_U32U32U64***
Function	Multiply unsigned 32bits by unsigned 32 bits result unsigned 64 bits
Input values	**R3:R2:R1:R0** 1st operand **R7:R6:R5:R4** 2nd operand
Output values	**R15:R14:R13:R12:R11:R10:R9:R8** Product
Destroy	Flags,R0,R1,R2,R3,R4,R5,R6,R7
Time	Min=566 Max=757 clocks
Observations	---

Example	Multiply 12345678 by 87654321
	
	Ldial 12345678
	Mov r0,**Acc**
	Mov r1**,AccH**
	Mov r2,**AccT**
	Mov r3,**AccTH**
	Ldial 87654321
	Mov r4,**Acc**
	Mov r5**,AccH**
	Mov r6,**AccT**
	Mov r7,**AccTH**
	rcall _MUL_U32U32U64
	Call _MUL_U32U32U64 (chips>=16k)

MUL_U32U32X10 (MUL_U32U32X10.INC File)

Description

Multiply unsigned 32bits number by 10 result back unsigned 32 bits

Implemented Functions

Name	***_MUL_U32U32X10***
Function	Multiply unsigned 16bits number by 10 result back unsigned 16 bits
Input values	**R3:R2:R1:R0** 1st operand
Output values	**R3:R2:R1:R0** Result
Destroy	Flags,R4,R5,R6,R7
Time	Max=24 clocks
Observations	---

Example	Multiply 12345678 by 10
	
	Ldial 12345678
	Mov r0,**Acc**
	Mov r1,**AccH**
	Mov r2,**AccT**
	Mov r3,**AccTH**
	
	rcall _MUL_U32U32X10
	Call _MUL_U32U32X10 (chips>=16k)

MULDIV_U8U8U8 (MULDIV_U8U8U8.INC File)

Description

Multiply unsigned 8bits by unsigned 8bits and divide product by unsigned 8 bits. The orders of operations are A by B then divide by C, this useful for multiply a value by a fraction.

$$D = \frac{A.B}{C}$$

Implemented Functions

Name	***_MULDIV_U8U8U8***
Function	Multiply unsigned 8bits by unsigned 8bits and product is divided by unsigned 8bits
Input values	**R0** A value **R1** B value **R2** C value
Output values	**R1:R0** D Quotient **R5:R4** D Remainter
Destroy	Flags,R2,R3
Time	Min=278 Max=318 clocks
Observations	---

Example	Multiply 100 by 5/4
	
	Ldi **Acc**,100
	Mov r0,**Acc**
	Ldi **Acc**,5
	Mov r1,**Acc**
	Ldi **Acc**,4
	Mov r2,**Acc**
	
	rcall _MULDIV_U8U8U8
	Call _MULDIV_U8U8U8 (chips>=16k)

UBTOS (UBTOS.INC File)

Description

Convert unsigned byte to string ASCII

Implemented Functions

Name	***_UBTOS***
Function	Convert unsigned byte to string ASCII automatic length
Input values	**Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 12 to string Ldi **Acc**,12 rcall _UBTOS Call _UBTOS (chips>=16k) Result "12\0"

Name	***_UBTOSU***
Function	Convert unsigned byte to string ASCII fixed 4 bytes length with format +nnn/0
Input values	**Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 12 to string Ldi **Acc**,12 rcall _UBTOSU Call _UBTOSU (chips>=16k) Result "+012\0"

Name	***_UBTOSUS***
Function	Convert unsigned byte to string ASCII fixed 4 bytes length with format +nnn/0
Input values	**Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 12 to string Ldi **Acc**,12 rcall _UBTOSUS Call _UBTOSUS (chips>=16k) Result "+ 12\0"

UITOH (UITOH.INC File)

Description

Convert integer 16bits to 4digits HEX string ASCII .

Implemented Functions

Name	***_UITOH***
Function	Convert integer 16bits to 4digits HEX string ASCII .
Input values	**AccH:Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 1000 to hex string Ldiaw 1000 rcall _UITOH Call _UITOH (chips>=16k) Result "03E8\0"

UITOS (UBTOS.INC File)

Description

Convert unsigned integer 16bits to string ASCII

Implemented Functions

Name	***_UITOS***
Function	Convert unsigned integer 16bits to string ASCII automatic length
Input values	**AccH:Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 1234 to string Ldiaw 1234 rcall _UITOS Call _UITOS (chips>=16k) Result "1234\0"

Name	***_UITOSU***
Function	Convert unsigned byte to string ASCII fixed 6 bytes length with format +nnnnn/0
Input values	**AccH:Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 12 to string Ldiaw **Acc**,12 rcall _UITOSU Call _UITOSU (chips>=16k) Result "+00012\0"

Name	***_UITOSUS***
Function	Convert unsigned integer to string ASCII fixed 6 bytes length with format +nnnnn/0
Input values	**Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 12 to string Ldi **Acc**,12 rcall _UITOSUS Call _UITOSUS (chips>=16k) Result "+ 12\0"

ULTOH (ULTOH.INC File)

Description

Convert integer 32bits to 8digits HEX string ASCII .

Implemented Functions

Name	***_ULTOH***
Function	Convert integer 32bits to 8digits HEX string ASCII .
Input values	**AccTH:AccT:AccH:Acc** unsigned value
Output values	**Z->** SRAM string ASCII with zero ended
Destroy	Flags
Time	---
Observations	---
Example	Convert 1000 to hex string Ldial 1000 rcall _ULTOH Call _ULTOH (chips>=16k) Result “000003E8\0”

INTFRAC – Integer and Fractions

IF_MUBSFB (MUBSFB.INC)

Description

Multiply unsigned byte by signed byte fraction useful for analog digital processing.

Implemented Functions

Name	***_IF_MUL_UBSFB***
Function	Multiply unsigned byte by signed byte fraction
Input values	**R0** unsigned integer **R1** signed fraction
Output values	**R3:R2** signed product
Destroy	Flags,R0
Time	----
Observations	----

Example	Multiply 100 by 0.34
	
	Ldi **Acc**,100
	Mov r0,**Acc**
	Ldi **Acc**,34*128/100
	Mov r1,**Acc**
	rcall _IF_MUL_UBSFB
	Call _IF_MUL_UBSFB (chips>=16k)

IF_MUSFSI (MUSFSI.INC)

Description

Multiply signed integer by signed integer fraction useful for analog digital processing.

Implemented Functions

Name	***_IF_MUL_SFSI***
Function	Multiply signed integer by signed integer fraction
Input values	**R1:R0** signed fraction number **R3:R2** signed integer number
Output values	**R5:R4** signed integer product
Destroy	Flags,R0,R1,R2,R3,R6
Time	Min=156 Max=187
Observations	----

Example	Multiply 100 by 0.34
	
	Ldiaw 100
	Mov r0,**Acc**
	Mov r1,**AccH**
	Ldiaw 34*32768/10000
	Mov r2,**Acc**
	Mov r3,**AccH**
	rcall _IF_MUL_SFSI
	Call _IF_MUL_SFSI (chips>=16k)

LOGIC

PARITY (PARITY.INC)

Description

Check if byte parity is odd or even.

Implemented Functions

Name	***_INTEGER_PARITY8***
Function	Check if byte parity is odd or even
Input values	**R0** input value
Output values	**Cy=1** if odd else even
Destroy	Flags,R1
Time	Max 22 clocks
Observations	----

Example	Check parity of **Acc**=3 "even"
	
	Ldi **Acc**,3
	Mov r0,**Acc**
	rcall _INTEGER_PARITY8
	Call _INTEGER_PARITY8 (chips>=16k)
	
	Result CY=0

ROTATION

ROTATED2D (ROTATION2D.INC)

Description

Perform a 16bits rotation of **P(x,y)** point in relation at 16bits **C(x,y)** center pointer by angle **A.** This is performed using bellow trigonometrics reduction.

(1) rotation of point **P(px,py)** with center at **P(cx,cy)** by $\boldsymbol{\alpha}$ angle.

$$r = \sqrt{(px - cx)^2 + (py - cy)^2},\ r = distance\ betwwen\ points$$

$$\alpha = atn2pts(cx, cy, px, py), \qquad angle\ between\ p(px, py) and\ p(cx, cy)$$

$$npx = cx + r.\cos(\alpha + \beta)\,, rotation\ of\ x\ axis\ by\ \beta\ angle$$
$$npy = cy + r.\sin(\alpha + \beta)\,, rotation\ of\ y\ axis\ by\ \beta\ angle$$

Consider **a,b,c** side of triangle rectangle, $\boldsymbol{\alpha}$ a opposite angle of side **a**, center pointer at intersection of side **c** and **a**, and desiderate point to be rotated at intersection of side **a** and **c**, yields:

$$a = py - cy,\ \ b = px - cx,\ \ \sin(\alpha) = \frac{a}{c}$$
$$c = \sqrt{a^2 + b^2}, \cos(\alpha) = \frac{b}{c}$$

Using a law of sun of sines and cosines, where k is rotate angle follow

$$\sin(\alpha + k) = \sin(\alpha).\cos(k) + \sin(k).\cos\,(\alpha)$$
$$\cos(\alpha + k) = \cos(\alpha).\cos(k) - \sin(a).\sin\,(k)$$

Replacing by equation (1), and have in mind that cos(k) and sin(k) area constants then, k=cos(k) and j=sin(k)

$$\sin(\alpha + k) = \frac{a}{c}.k + \frac{b}{c}.j$$

$$\cos(\alpha + k) = \frac{b}{c}.k - \frac{a}{c}.j$$

Knowing that $c = \sqrt{a^2 + b^2}$ represent a distance between points we rewrite rotation equation as follow

$$npx = cx + c.\left(\frac{b}{c}.k - \frac{a}{c}.j\right), rotation\ of\ x\ axis\ by\ k\ angle$$
$$npy = cy + c.\left(\frac{a}{c}.k + \frac{b}{c}.j\right), rotation\ of\ y\ axis\ by\ k\ angle$$

Cutting **c**=distance, yields

$$npx = cx + (b.k - a.j)$$
$$npy = cy + (a.k + b.j)$$

That manner we obtain a very fast routine because only need to realize is a table lookup of sine of desiderate request angle and 4 integer multiplies.

Implemented Functions

Name	***_ROT2D_SET_POINT***
Function	Set coordinates of point to be rotated
Input values	**X,Y** point coordinates
Output values	None
Destroy	None
Time	----
Observations	----

Example	Set point P(100,150) to be rotated
	
	Ldiw **X**,100
	Ldiw **Y**,150
	rcall _ROT2D_SET_POINT
	Call _ROT2D_SET_POINT (chips>=16k)

Name	***_ROT2D_GET_POINT***
Function	Get coordinates of point to be rotated
Input values	None
Output values	**X,Y** point coordinates
Destroy	None
Time	----
Observations	----
Example	Get point to be rotated, after this **X,Y** will have the coordinates rcall _ROT2D_GET_POINT Call _ROT2D_GET_POINT (chips>=16k)

Name	***_ROT2D_SET_CENTER_POINT***

Function	Set coordinates of center of rotation
Input values	**X,Y** center point coordiantes
Output values	None
Destroy	None
Time	----
Observations	----

Example	Set Center point P(100,150)
	
	Ldiw X,100
	Ldiw Y,150
	rcall _ROT2D_SET_CENTER_POINT
	Call _ROT2D_SET_CENTER_POINT (chips>=16k)

Name	***_ROT2D_GET_CENTER_POINT***
Function	Get coordinates of center point
Input values	None
Output values	**X,Y** point coordinates
Destroy	None
Time	----
Observations	----
Example	Get center point, after this **X,Y** will have the coordinates rcall _ROT2D_GET_CENTER_POINT Call _ROT2D_GET_CENTER_POINT (chips>=16k)

Name	***_ROT2D_GET_ROTATED_POINT***
Function	Get coordinates of rotated point
Input values	None
Output values	**X,Y** point coordinates
Destroy	None
Time	----
Observations	----
Example	Get coordinates of rotated point, after this **X,Y** will have the coordinates rcall _ROT2D_GET_ROTATED_POINT Call _ROT2D_GET_ROTATED_POINT (chips>=16k)

Name	_ROT2D_SET_ANGLE
Function	Set angle of rotation
Input values	**AccH:Acc** angle in degree
Output values	None
Destroy	None
Time	----
Observations	----
Example	Set angle of rotation to 27 degree rcall _ROT2D_SET_ANGLE Call _ROT2D_GET_ANGLE (chips>=16k)

Name	_ROT2D_GET_ANGLE
Function	Get angle of rotation
Input values	None
Output values	**AccH:Acc** angle in degree
Destroy	None
Time	----
Observations	----
Example	Get angle of rotation rcall _ROT2D_GET_ANGLE Call _ROT2D_GET_ANGLE (chips>=16k)

Name	_ROT2D_ROTATE
Function	Rotate desiderate p(x,y) in relation of center(x,y) by angle alfa
Input values	None
Output values	None
Destroy	Flags,R0..R15
Time	----
Observations	----

Example	Rotate point p(100,128) at center(40,30) by angle 50 degree.
	Ldiaw 50
	Call _ROT2D_SET_ANGLE
	Ldiw X,100
	Ldiw Y,128
	Call _ROT2D_SET_POINT
	Ldiw X,40
	Ldiw Y,30
	Call _ROT2D_SET_CENTER_POINT
	Call _ROT2D_ROTATE
	Call _ROT2D_ROTATED_POINT

TRIGONOMETRY

DISCRETE COSINE (DISCRETE_COSINE.INC)

Description

Calculate a integer 16bits discrete and scaled cosine.

Name	***_DSCOS16B***
Function	Compute scaled cosine using a below equation $$y = s.\cos(\alpha)$$ Where y=signed integer scaled cosine function A=signed integer angle S=unsigned integer scale
Input values	**AccH:Acc** angle in degree **AccTH:AccT** unsigned scale factor
Output values	**AccH:Acc** signed integer scaled cosine of angle A
Destroy	Flags,R0,R1,R2,R3,R4,R5,R6
Time	----
Observations	----
Example	Compute cosine of 100 degrees and scale it by a factor of 50 Ldiaw 100 Ldiawt 50 rcall _DSCOS16B Call _DSCOS16B (chips>=16k)

Name	_DCOS16B
Function	Compute discrete integer cosine
Input values	**AccH:Acc** angle in degree
Output values	**AccH:Acc** signed integer cosine of angle A
Destroy	Flags,R0
Time	----
Observations	Output format of 16bits value of sine

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	i	f	f	f	f	f	f	f	f	f	f	f	f	f	f

s=signal 0=positive 1=negative
i=integer part
f=fractionary part

Example

Compute cosine of 100 degrees

```
Ldiaw 100
rcall _DCOS16B
Call  _DCOS16B (chips>=16k)
```

DISCRETE SINE (DISCRETE_SINE.INC)

Description

Calculate a integer 16bits discrete and scaled sine.

Name	***_DSSIN16B***
Function	Compute scaled sine using a below equation $$y = s.\sin(\alpha)$$ Where y=signed integer scaled sine function A=signed integer angle S=unsigned integer scale
Input values	**AccH:Acc** angle in degree **AccTH:AccT** unsigned scale factor
Output values	**AccH:Acc** signed integer scaled sine of angle A
Destroy	Flags,R0,R1,R2,R3,R4,R5,R6
Time	----
Observations	----
Example	Compute sine of 100 degrees and scale it by a factor of 50 Ldiaw 100 Ldiawt 50 rcall _DSSIN16B Call _DSSIN16B (chips>=16k)

Name	***_DSIN16B***
Function	Compute discrete integer cosine
Input values	**AccH:Acc** angle in degree
Output values	**AccH:Acc** signed integer cosine of angle A
Destroy	Flags,R0
Time	----
Observations	Output format of 16bits value of sine 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 s i f f f f f f f f f f f f f f s=signal 0=positive 1=negative i=integer part f=fractionary part
Example	Compute sine of 100 degrees Ldiaw 100 rcall _DSIN16B Call _DSIN16B (chips>=16k)

DISPLAYS

HD44780

VERSION10 - Deprecated

VERSION20 (DPDRV48B.INC)

Description

The HD44780U dot-matrix liquid crystal display controller and driver LSI displays alphanumerics, Japanese kana characters, and symbols. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor. Since all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver.

A single HD44780U can display up to one 8-character line or two 8-character lines. The HD44780U has pin function compatibility with the HD44780S which allows the user to easily replace an LCD-II with an HD44780U. The HD44780U character generator ROM is extended to generate Implemented Functions.

Author Version20 drive for HD4470 has a complete set of functions to use in 4 or 8bits mode using 5x8 or 5x10 character fonts, capable to use data bits in any port(no memory mapped port) in any bit position and have possibility to set a port for data and command in same port or separately.

Implemented Functions

Name	**_DISP_INIT**
Function	Initialize HD44780 Interface
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----

Example	Initialize HD44780 with DATA and COMMAND in same port in this case PORTC, RS bit=4,RW bit=5, E bit=6, two(2) lines, 5x8 font, interface 4 bits, interface bit start position 0.
	
	`.EQU _DISP_PORT_DATA_OUT = PORTC`
	`.EQU _DISP_PORT_DATA_IN = PINC`
	`.EQU _DISP_PORT_DATA_DIR = DDRC`
	
	`.EQU _DISP_PORT_CMD_OUT = PORTC`
	`.EQU _DISP_PORT_CMD_IN = PINC`
	`.EQU _DISP_PORT_CMD_DIR = DDRC`
	
	`.EQU _DISP_BIT_RS = 4`
	`.EQU _DISP_BIT_RW = 5`
	`.EQU _DISP_BIT_E = 6`
	
	`.EQU _DISP_LINE = _DISP_LINE_2`
	`.EQU _DISP_FONT = _DISP_FONT_5X8`
	`.EQU _DISP_INTERFACE = _DISP_4BITS`
	`.EQU _DISP_POSITION = 0`
	
	`rcall _DISP_INIT`
	`call _DISP_INIT (chips >=16k)`

Name	_DISP_CMD_WRITE
Function	Write a COMMAND in HD44780
Input values	Acc Command to be write
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Sent a command to clear display. Ldi Acc,_DISP_CMD_CLEAR rcall _DISP_CMD_WRITE call _DISP_CMD_WRITE (chips >=16k)

Name	_DISP_DATA_WRITE
Function	Write a DATA in HD44780
Input values	Acc Data to be write
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Sent a '*' to display. Ldi Acc,'*' rcall _DISP_DATA_WRITE call _DISP_DATA_WRITE (chips >=16k)

Name	_DISP_SEND_STR
Function	Send a string in FLASH to HD44780
Input values	Z-> String in Flash
Output values	None
Destroy	Flags,R0
Time	----
Observations	----
Example	Sent a "Hello World" to display. Ldiw Z,MSG*2 Rcall _DISP_SEND_STR call _DISP_SEND_STR (chips >=16k) . . MSG: .DB "Hello World",0

Name	_DISP_SEND_STR_S
Function	Send a string in SRAM to HD44780
Input values	**Z->** String in SRAM
Output values	None
Destroy	Flags,R0
Time	----
Observations	----
Example	Sent a "Hello World" to display. Ldiw Z,MSG Rcall _DISP_SEND_STR_S call _DISP_SEND_STR_S (chips >=16k) . Memory position MSG must filled with "Hello World",0 message before execute above code .DSEG MSG: .BYTE 11 .CSEG

Name	_DISP_LOCATE
Function	Set cursor position in HD44780
Input values	**Acc** Line **AccH** Column
Output values	None
Destroy	Flags
Time	----
Observations	----

Example	Put display cursor at line 2 column 5.
	
	Ldi Acc,2
	Ldi AccH,5
	Rcall _DISP_LOCATE
	call _DISP_LOCATE (chips >=16k)

Name	_DISP_CLEAR
Function	Clear entire display screen and position cursor at line 1 column 1
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Rcall _DISP_CLEAR call _DISP_CLEAR (chips >=16k)

Name	_DISP_HOME
Function	Put cursor at line 1 column 1
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Rcall _DISP_HOME call _DISP_HOME (chips >=16k)

Name	_DISP_DATA_READ
Function	Read contents of last cursor position from CGRAM or DDRAM
Input values	None
Output values	**Acc** Data read
Destroy	Flags
Time	----
Observations	----
Example	Rcall _DISP_DATA_READ call _DISP_DATA_READ (chips >=16k)

Name	***_DISP_DISPLAY***
Function	Turn display character on screen visible or not
Input values	**Acc** _ON= Visible _OFF=hidden
Output values	None
Destroy	Flags
Time	----
Observations	----

Example	Turn display visible and hidden
	Ldi **Acc**,_ON
	Rcall _DISP_DISPLAY
	Ldi **Acc**,_OFF
	call _DISP_DISPLAY (chips >=16k)

Name	***_DISP_SCROLL_LEFT***
Function	Scroll entire display to left
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	rcall _DISP_SCROLL_LEFT call _DISP_SCROLL_LEFT (chips >=16k)

Name	***_DISP_SCROLL_RIGHT***
Function	Scroll entire display to right
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	rcall _DISP_SCROLL_RIGHT call _DISP_SCROLL_RIGHT (chips >=16k)

Name	_DISP_CURSOR
Function	Turn cursor on or off
Input values	Acc _ON or _OFF
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Turn cursor off Ldi Acc,_OFF rcall _DISP_CURSOR call _DISP_CURSOR (chips >=16k)

Name	_DISP_CURSOR_LEFT
Function	Move cursor to the left one position
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	rcall _DISP_CURSOR_LEFT call _DISP_CURSOR_LEFT (chips >=16k)

Name	_DISP_CURSOR_RIGHT
Function	Move cursor to the right one position
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	rcall _DISP_CURSOR_RIGHT call _DISP_CURSOR_RIGHT (chips >=16k)

Name	_DISP_BLINK
Function	Turn cursor blink state on or off
Input values	Acc _ON or _OFF
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Turn cursor blink off Ldi Acc,_OFF rcall _DISP_BLINK call _DISP_BLINK (chips >=16k)

Name	_DISP_SET_CGRAM_ADDR
Function	Set address for further data write/read in CGRAM area
Input values	Acc address
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Set CGRAM address to 5 Ldi Acc,5 rcall _DISP_SET_CGRAM_ADDR call _DISP_SET_CGRAM_ADDR (chips >=16k)

Name	_DISP_SET_DDRAM_ADDR
Function	Set address for further data write/read in DDRAM area
Input values	Acc address
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Set DRAM address to 5 Ldi Acc,5 rcall _DISP_SET_DDRAM_ADDR call _DISP_SET_DDRAM_ADDR (chips >=16k)

Name	_DISP_REDEFINE_CHAR
Function	Redefine character pattern for display codes 0..7
Input values	**Acc** character code 0..7 **Z->** Flash area with new character pattern
Output values	None
Destroy	Flags
Time	----
Observations	----

Example	Set a box pattern to character 2
	
	Ldi **Acc**,2
	Ldiw Z,BOX_PATTERN
	rcall _DISP_SET_DDRAM_ADDR
	call _DISP_SET_DDRAM_ADDR (chips >=16k)
	
	BOX_PATTERN: .DB 0x1f,0x11,0x11,0x11,0x11,0x11,0x1f,0x00

HD44780 Constants		
Name	**Value**	**Description**
_DISP_LINE_1	0	1 LINE
_DISP_LINE_2	1	2 LINE
_DISP_FONT_5X8	0	5X8 FONT SIZE
_DISP_FONT_5X10	1	5X10 FONT SIZE
_DISP_4BITS	0	4 bits interface
_DISP_8BITS	1	8 bits interface

BIG_NUMBERS (BIG_NUMBER.INC)

Description

Display Big Digits in displays with 2 lines, these digits use the bellow pattern.

```
 +- -+     | -- -+ -- -+ |   | +- -- +-    -- -+ +- -+ +- -+
 |1 2| '' 3|  6 4| 6  4| |L 0| |5 6  |1 ''  7 2| |5 4| |5 4|
 |   |     | -- -+ -- -+ +- -+ +- -- |         | +- -+ +- -+
 |   |     | |         |     |     | +- -+     | |   |     |
 |L 0| '' 3| |L  _  _ 0| '' 3|  - 0| |5 4| '' 3| |L 0| '' 3|
 +- -+     | +- -- -- -+     | -- -+ +- -+     | +- -+     |
```

The above number represent characters codes 0..7,'-`,'L',' `, codes 0..7 was redefined during drive initialization.

Name	***_DISP_BIG_NUMBER_INIT***
Function	Initialize BIG NUMBERS redefining character pattern for display codes 0..7
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	Need HD44780 Version20 display drive
Example	rcall _DISP_SET_DDRAM_ADDR call _DISP_SET_DDRAM_ADDR (chips >=16k)

Name	***_DISP_PRINT_BIG_NUMBER***
Function	Put big number on display screen
Input values	**Acc** Number in ASCII
Output values	None
Destroy	Flags
Time	----
Observations	Need HD44780 Version20 display drive

Example	Put big number '3' at column 5
	
	Ldi **Acc**,'3'
	Ldi **AccH**,5
	rcall _DISP_PRINT_BIG_NUMBER
	call _DISP_PRINT_BIG_NUMBER (chips >=16k)

SED1335

SED1335 (G320X240.INC)

Description

The SED1330/1335F/1336F is a family of versatile LCD controller ICs that can display text and graphics on a medium size LCD panel. The software is compatible among all three chips. S-MOS recommends new designs use the SED1335 since the SED1330 will gradually be replaced by the SED1335.

Features

- Text, graphics and combined text/graphics displaymodes
- Three overlapping screens in graphics mode
- 640 ´ 256 pixel LCD panel display resolution
- Programmable cursor control
- Smooth horizontal and vertical scrolling of all or part of the display
- 1/2-duty to 1/256-duty LCD drive
- Up to 64 Kbytes of external static RAM frame buffer memory
- Internal character generator
- 160, 5 ´ 7 pixel characters in internal maskprogrammed character generator ROM
- Up to 64, 8 ´ 16 pixel characters in external character generator RAM
- Up to 256, 8 ´ 16 pixel characters in external character generator ROM
- 6800 and 8080 family microprocessor interfaces

Author Version20 drive for SED1335 has a complete set of functions capable to use data bits in any port(no memory mapped port) have possibility to set a port for data and command in same port or separately.

Implemented Functions

Name	***_DISP_INIT***
Function	Initialize SED1335 Interface
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----

Example	Initialize SET1335 with DATA port in PORTA, COMMAND port in PORTB, WR bit=7,RD bit=6, CS bit=5, A0 bit=4, RESET bit=3.
	
	`.EQU _DISP_PORT_DATA_OUT = PORTA`
	`.EQU _DISP_PORT_DATA_IN = PINA`
	`.EQU _DISP_PORT_DATA_DIR = DDRA`
	
	`.EQU _DISP_PORT_CMD_OUT = PORTB`
	`.EQU _DISP_PORT_CMD_IN = PINB`
	`.EQU _DISP_PORT_CMD_DIR = DDRB`
	
	`.EQU _DISP_BIT_WR = 7`
	`.EQU _DISP_BIT_RD = 6`
	`.EQU _DISP_BIT_CS = 5`
	`.EQU _DISP_BIT_A0 = 4`
	`.EQU _DISP_BIT_RESET = 3`
	
	`rcall _DISP_INIT`
	`call _DISP_INIT (chips >=16k)`

Name	_DISP_FILL
Function	Fill VRAM area with specific pattern
Input values	X Stard Addres of VRAM Y Number of bytes Acc Pattern
Output values	None
Destroy	Flags
Time	----
Observations	----

Example	Fill VRAM area start at 0 length of 1000 bytes with patter 0xaa
	
	Ldiw X,0
	Ldiw Y,1000
	Ldi Acc,0xaa
	rcall _DISP_FILL
	call _DISP_FILL (chips >=16k)

Name	_DISP_SET_CURSOR_ADDR
Function	Set Cursor Address
Input values	AccH:Acc Cursor Address
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Set cursor address to 1000H. Ldiaw 0x1000 rcall _DISP_SET_CURSOR_ADDR call _DISP_SET_CURSOR_ADDR (chips >=16k)

Name	_DISP_GET_CURSOR_ADDR
Function	get Cursor Address
Input values	None
Output values	AccH:Acc Cursor Address
Destroy	Flags
Time	----
Observations	----
Example	rcall _DISP_GET_CURSOR_ADDR call _DISP_GET_CURSOR_ADDR (chips >=16k)

Name	_DISP_SEED
Function	Define interface speed
Input values	Acc _DP_SLOW or _DP_FAST
Output values	None
Destroy	None
Time	----
Observations	SED1335 GENERATE A FLICKS ON SCREEN AT EACH COMMAND OR DATA RECEIVED , WHEN USER DEFINE A SLOW SPEED INTERFACE LESS FLICKS THEN GENERATE ON SCREEN
Example	Set interface speed to fast Ldi Acc,_DP_FAST rcall _DISP_SPEED call _DISP_SPEED (chips >=16k)

Name	_DISP_CMD
Function	Send a COMMAND to display
Input values	Acc COMMAND
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Send a COMMAND to set cursor direction increment to right Ldi Acc,_DISP_CMD_CSRDIR_RIGHT rcall _DISP_CMD call _DISP_CMD (chips >=16k)

Name	_DISP_DATA
Function	Send a DATA to display
Input values	Acc DATA
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Send a '*' to display Ldi Acc,'*' rcall _DISP_DATA call _DISP_DATA (chips >=16k)

Name	_DISP_SET_LC
Function	Set cursor position on screen
Input values	**XL** column **YL** line
Output values	None
Destroy	Flags
Time	----
Observations	----

Example	Set cursor at line 2 column 5
	
	Ldi **XL**,5
	Ldi **YL**,2
	rcall _DISP_SET_LC
	call _DISP_SET_LC (chips >=16k)

Name	_DISP_SEND_STR
Function	Send a string in FLASH to display
Input values	**Z->** String in Flash
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Sent a "Hello World" to display. Ldiw Z,MSG*2 Rcall _DISP_SEND_STR call _DISP_SEND_STR (chips >=16k) . . MSG: .DB "Hello World",0

Name	_DISP_SEND_STR_S
Function	Send a string in SRAM to display
Input values	Z-> String in SRAM
Output values	None
Destroy	Flags,R0
Time	----
Observations	----
Example	Sent a "Hello World" to display. Ldiw Z,MSG Rcall _DISP_SEND_STR_S call _DISP_SEND_STR_S (chips >=16k) . Memory position MSG must filled with "Hello World",0 message before execute above code .DSEG MSG: .BYTE 11 .CSEG

Name	_DISP_PSET
Function	Set or Clear a pixel at display coordinates
Input values	X,Y Coordinates Acc _ON to set _OFF to clear
Output values	None
Destroy	Flags
Time	----
Observations	----

Example	Set pixel at coordinates (160,120)
	
	Ldiw X,160
	Ldiw Y,120
	Ldi Acc,_ON
	Rcall _DISP_PSET
	call _DISP_PSET (chips >=16k)

Name	_DISP_POINT
Function	Get pixel stated at display coordinates
Input values	**X,Y** Coordinates
Output values	**Acc** _ON if set _OFF if clear
Destroy	Flags
Time	----
Observations	----

Example	Get pixel state at coordinates (160,120)
	
	Ldiw **X**,160
	Ldiw **Y**,120
	Rcall _DISP_POINT
	call _DISP_POINT (chips >=16k)

SED1335 Constants		
Name	**Value**	**Description**
_DISP_WIDTH	320	Hardware display width
_DISP_HEIGHT	240	Hardware display height
_DISP_SCALE_WIDTH	320	Logical width
_DISP_SCALE_HEIGHT	240	Logical height
_DISP_CMD_SYSTEM_SET	0x40	Initialize display
_DISP_CMD_SLEEP_IN	0x53	Enter standby mode
_DISP_CMD_DISP_OFF	0x58	Disable display
_DISP_CMD_DISP_ON	0x59	Enable display
_DISP_CMD_SCROLL	0x44	Set display start Addr and regions
_DISP_CMD_CSRFORM	0x5d	Set cursor type
_DISP_CMD_CGRAM_ADR	0x5c	Set addr of character generator in RAM
_DISP_CMD_CSRDIR_RIGHT	0x4c	Set cursor movement to right
_DISP_CMD_CSRDIR_LEFT	0x4d	Set cursor movement to left
_DISP_CMD_CSRDIR_UP	0x4e	Set cursor movement to up
_DISP_CMD_CSRDIR_DOWN	0x4f	Set cursor movement to down
_DISP_CMD_HDOT_SCR	0x5a	Set Horizontal scroll position
_DISP_CMD_OVLAY	0x5b	Set display overlay format
_DISP_CMD_CSWR	0x46	Set cursor address
_DISP_CMD_CSRR	0x47	Read cursor address
_DISP_CMD_MWRITE	0x42	Write to display memory
_DISP_CMD_MREAD	0x43	Read to display memory
_DISP_BIT_BUSY	6	Internal bit busy flag

SEVEN

DRTYP1 (DIPS7DR1.INC)

Description

This drive is software generator scanning for seven segments display generally LED displays. It generate pattern for space, digits 0..9, and characters ‘A’ to ‘Z’ of course some character not possible to implement in seven segment but a close pattern is generated as follow.

```
SP   0    1    2    3    4    5    6    7    8    9
 --   ##   --   ##   ##   --   ##   ##   ##   ##   ##
|  | #  # |  # |  # |  # #  # #  | #  | |  # #  # #  #
|  | #  # |  # |  # |  # #  # #  | #  | |  # #  # #  #
 --   --   --   ##   ##   ##   ##   ##   --   ##   ##
|  | #  # |  # #  | |  # |  # |  # #  # |  # #  # |  #
|  | #  # |  # #  | |  # |  # |  # #  # |  # #  # |  #
 --   ##   --   ##   ##   --   ##   ##   --   ##   --

 A   B    C    D    E    F    G    H    I    J    L
 ##   --   ##   --   ##   ##   ##   --   --   --   --
#  # #  | #  | |  # #  | #  | #  # #  # |  # |  # #  |
#  # #  | #  | |  # #  | #  | #  # #  # |  # |  # #  |
 ##   ##   --   ##   ##   ##   ##   ##   --   --   --
#  # #  # #  | #  # #  | #  | |  # #  # |  # #  # #  |
#  # #  # #  | #  # #  | #  | |  # #  # |  # #  # #  |
 --   ##   ##   ##   ##   --   ##   --   --   ##   ##

 N    O    P    Q    R    S    U    Y    V    Z
 ##   ##   ##   ##   ##   ##   --   --   ##   ##
#  # #  # #  # #  # #  # #  | #  # #  # |  # |  #
#  # #  # #  # #  # #  # #  | #  # #  # |  # |  #
 --   --   ##   ##   --   ##   --   ##   --   ##
#  # #  # #  | |  # #  | |  # #  # |  # |  # #  |
#  # #  # #  | |  # #  | |  # #  # |  # |  # #  |
 --   ##   --   --   --   ##   ##   --   --   ##
```

```
      DISPLAY MATRIX CONFIGURATION

           BIT_DIG_7  DIT_DIG_6           BIT_DIG_0

               |            |                  |
               |            |                  |
             a |          a |                a |
            -------      -------            -------
SEG_A >--  |       |    |       |          |       |
SEG_B >--  |       |    |       |          |       |
SEG_C >-- f|       |b  f|       |b        f|       |b
SEG_D >--  |       |    |       |          |       |
SEG_E >--  |   g   |    |   g   |          |   g   |
SEG_F >--   -------      -------   .......  -------
SEG_G >--  |       |    |       |          |       |
SEG_P >--  |       |    |       |          |       |
          e|       |c  e|       |c        e|       |c
           |       |    |       |          |       |
           |       |    |       |          |       |
            ------- O P  ------- O P        -------  O P
               d            d                  d
```

Implemented Functions

Name	***_DISP7_INIT***
Function	Initialize Seven segment interface
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	Global interrupts are disable during initialization
Example	Initialize using PORTB as segment,PORTD as Digits, Segment ON=LOW,Segment OFF=High,Digit ON=low, Digit OFF=High,Number of digits=8, segments bit number of A,B,C,E,F,G,P in sequence 0,1,2,3,4,5,6,7 and Digits bit numbers of 0,1,2,3,4,5,6,7 as same in sequence. .EQU _DISP7_SEG_PORT_OUT = PORTB .EQU _DISP7_SEG_PORT_DIR = DDRB .EQU _DISP7_SEG_PORT_IN = PINB .EQU _DISP7_DIG_PORT_OUT = PORTD .EQU _DISP7_DIG_PORT_DIR = DDRD .EQU _DISP7_DIG_PORT_IN = PIND .EQU _DISP7_SEG_ON = 0 .EQU _DISP7_SEG_OFF = 1 .EQU _DISP7_DIG_ON = 0 .EQU _DISP7_DIG_OFF = 1 .EQU _DISP7_NUM_DIGITS = 8 .EQU _DISP7_BIT_SEG_A = 0

	.EQU _DISP7_BIT_SEG_B = 1 .EQU _DISP7_BIT_SEG_C = 2 .EQU _DISP7_BIT_SEG_D = 3 .EQU _DISP7_BIT_SEG_E = 4 .EQU _DISP7_BIT_SEG_F = 5 .EQU _DISP7_BIT_SEG_G = 6 .EQU _DISP7_BIT_SEG_P = 7 .EQU _DISP7_BIT_DIG_0 = 0 .EQU _DISP7_BIT_DIG_1 = 1 .EQU _DISP7_BIT_DIG_2 = 2 .EQU _DISP7_BIT_DIG_3 = 3 .EQU _DISP7_BIT_DIG_4 = 4 .EQU _DISP7_BIT_DIG_5 = 5 .EQU _DISP7_BIT_DIG_6 = 6 .EQU _DISP7_BIT_DIG_7 = 7 rcall _DISP7_INIT call _DISP7_INIT (chips >=16k)

Name	***_DISP7_SHOW_DIGIT***
Function	Turn on one digit a time
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	IT'S RECOMMEND THAT PROGRAMMER TO USE THIS ROUTINE IN THE INTERRUPT HANDLE IN FIXED TIME AT 160Hz MINIMUM FREQUENCY.

Example	Using this without a interrupt routine
	
	LOOP:
	      Call _DISP7_SHOW_DIGIT
	      'INSERT CODE HERE TO PROCESS OTHER THINGS
	      'SINCHRONIZE TO FIXED RATE
	RJM LOOP

Name	_DISP7_DATA
Function	Send a character to display buffer
Input values	**Acc** Character ASCII
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Show 'A' on display Ldi **Acc**,'A' Rcall _DISP7_DATA call _DISP7_DATA (chips >=16k)

Name	_DISP7_SEND_STR
Function	Send String to display
Input values	**Z→** Flash string with zero ended \0
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Show 'ABCDEF' on display Ldiw Z,MSG*2 Rcall _DISP7_SEND_STR call _DISP7_SEND_STR (chips >=16k) MSG: .DB "ABCDEF",0

Name	_DISP7_LOCATE
Function	Set cursor position
Input values	**AccH** Column
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Position cursor at column 4 Ldi **Acc**,4 Rcall _DISP7_LOCATE call _DISP7_LOCATE (chips >=16k)

Name	_DISP7_CLS
Function	Clear character buffer
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Rcall _DISP7_CLS call _DISP7_CLS (chips >=16k)

GRAPH

CIRCLE (CIRCLE.INC)

Description

Draw a Circle in any graph device that has implement a _DISP_PSET routine.

Implemented Functions

Name	**_GRAPH_CIRCLE**
Function	Draw a circle at any graph interface
Input values	**X,Y** Circle coordinates **AccTH:AccT** radius **Acc** Pixel color, if monochrome display _ON or _OFF
Output values	None
Destroy	None
Time	----
Observations	User must be define a routine called _DISP_PSET that receiver **X,Y** with coordinates parameters and **Acc** with color.

Example	Draw circle at coordinates (128,102) with radius=100
	
	Ldiw X,128
	Ldiw Y,102
	Ldiawt 100
	Ldi **Acc**,_ON
	rcall _GRAPH_CIRCLE
	call _GRAPH_CIRCLE (chips >=16k)

LINE (LINE.INC)

Description

Draw a line in any graph device that has implement a _DISP_PSET routine that receiver **X,Y** register with coordinates and **Acc** with pixel color.

Implemented Functions

Name	**_GRAPH_MOVE_TO**
Function	Move a virtual pen to specific coordinate that represent a initial line coordinate
Input values	**AccH:Acc** **X** coordinate **AccTH:AccT** **Y** coordinate
Output values	None
Destroy	None
Time	----
Observations	----
Example	Move graph cursor to coordinates (10,20) Ldiaw 10 Ldiawt 20 rcall _GRAPH_MOVE_TO call _GRAPH_MOVE_TO (chips >=16k)

Name	**_GRAPH_MOVE_TO_EX**
Function	Same as _GRAPH_MOVE_TO but use **X,Y** register instead
Input values	**X,Y** Coordinates to move
Output values	None
Destroy	None
Time	----
Observations	----

Example	Move graph cursor to coordinates (10,20)
	
	Ldiw **X**,10
	Ldiw **Y**,20
	rcall _GRAPH_MOVE_TO_EX
	call _GRAPH_MOVE_TO_EX (chips >=16k)

Name	_GRAPH_GET_POINT
Function	Get coordinates of virtual pen
Input values	None
Output values	**X,Y** Coordinates of pen
Destroy	None
Time	----
Observations	----
Example	rcall _GRAPH_GET_POINT call _GRAPH_GET_POINT (chips >=16k)

Name	_GRAPH_LINE_TO
Function	Draw a line any device from virtual pen coordinates to coordinates provide in this routine
Input values	X,Y End Coordinates
Output values	Flags,r0..r13
Destroy	None
Time	----
Observations	----

Example	Draw a line from coordinates (10,20) to coordinates (150,200)
	
	Ldiw X,10
	Ldiw y,20
	Call _GRAPH_MOVE_TO_EX
	Ldiw x,150
	Ldiw Y,200
	Ldi Acc,_ON
	Call _GRAPH_LINE_TO

DRAW (DRAW.INC)

Description

Draw is similar a DRAW command used in some BASIC languages that allow graphics drawing using a vectors defined by a string using ASCII characters like commands. Below as list of these commands.

Ortogonal moves

```
nnnU - draw a vector up    ( 90o)
nnnR - draw a vector right (  0o)
nnnD - draw a vector down  (270o)
nnnL - draw a vector left  (180o)
```

Diagonal moves

```
nnnE - draw a vector up and right   ( 45o)
nnnF - draw a vector down and right (315o)
nnnG - draw a vector down and left  (227o)
nnnH - draw a vector up and left    (135o)
```

Free moves

```
sxxx,syyyP - draw a vector to PX+xxx,PY+yyy
```

Pen control

```
W - Turn pen on
B - Turn pen off
```

Where nnn,xxx,yyy is a values ranging 0 to 255 meaning vector length, the below string draw on device a word “DRAW”.

"10U4R2F6D2G4LB8RW10U4R2F2D2G4LB4RW4FB2RW8U2E2R2F4D6LB6RW4DB2R10UW10D5E5F10U"

Implemented Functions

Name	_DRAW_SET_POINT
Function	Define a start point of vectors
Input values	X,Y Start point Coordinates
Output values	None
Destroy	None
Time	----
Observations	----

Example	Set start coordinate at (0,0)
	Ldiw X,0
	Ldiw Y,0
	rcall _DRAW_SET_POINT
	call _DRAW_SET_POINT (chips >=16k)

Name	_DRAW_GET_POINT
Function	Get a start point of vectors
Input values	None
Output values	X,Y Start point Coordinates
Destroy	None
Time	----
Observations	----
Example	rcall _DRAW_GET_POINT call _DRAW_GET_POINT (chips >=16k)

Name	_DRAW_SET_CENTER_POINT
Function	Define a center point for vectors rotation
Input values	X,Y center point Coordinates
Output values	None
Destroy	None
Time	----
Observations	----

Example	Set center rotate point coordinate at (100,200)
	Ldiw X,100
	Ldiw Y,200
	rcall _DRAW_SET_CENTER_POINT
	call _DRAW_SET_CENTER_POINT (chips >=16k)

Name	_DRAW_GET_CENTER_POINT
Function	Get a center point of vectors rotation
Input values	None
Output values	**X,Y** Start point Coordinates
Destroy	None
Time	----
Observations	----
Example	rcall _DRAW_GET_CENTER_POINT call _DRAW_GET_POINT (chips >=16k)

Name	_DRAW_SET_SCALE
Function	Set scale of vectors
Input values	**AccH:Acc AccH** integer part,**Acc** fractinary part
Output values	None
Destroy	None
Time	----
Observations	----
Example	Set scale to 2.5 Ldiaw (2+50/100)*256 rcall _DRAW_SET_SCALE call _DRAW_SET_SCALE (chips >=16k)

Name	_DRAW_GET_SCALE
Function	Get scale of vectors
Input values	None
Output values	**AccH:Acc** scale
Destroy	None
Time	----
Observations	----
Example	rcall _DRAW_GET_SCALE call _DRAW_GET_SCALE (chips >=16k)

Name	_DRAW_SET_ROTATE_ANGLE
Function	Set rotate angle of output set of vectors
Input values	AccH:Acc angle in degrees
Output values	None
Destroy	None
Time	----
Observations	----
Example	Set rotate angle to 50 Ldiaw 50 rcall _DRAW_SET_ROTATE_ANGLE call _DRAW_SET_ROTATE_ANGLE (chips >=16k)

Name	_DRAW_SAVE_POINT
Function	Save start point for further use
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	rcall _DRAW_SAVE_SCALE call _DRAW_SAVE_SCALE (chips >=16k)

Name	_DRAW_RESTORE_POINT
Function	Restore start point
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	rcall _DRAW_RESTORE_POINT call _DRAW_RESTORE_POINT(chips >=16k)

Name	**_DRAW**
Function	Draw set of vectors on device
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	Draw a word "DRAW" on graph device Ldiw Z,MSG*2 rcall _DRAW call _DRAW(chips >=16k) MSG: .db "10U4R2F6D2G4LB8RW10U4R2F2D2G4LB4RW4FB2RW8U2E2R2F4D6LB6RW4DB2R10UW10D5E5F10U",0

EEPROMS

AT24C64 (AT24C64.INC)

Description

The AT24C32/64 provides 32,768/65,536 bits of serial electrically erasable and programmable read only memory (EEPROM) organized as 4096/8192 words of 8 bits each. The device's cascadable feature allows up to 8 devices to share a common 2-wire bus.

Implemented Functions

Name	**_AT24C64_BYTE_WRITE**
Function	Write a byte into AT24C64 chip
Input values	**Acc** Data to be write **AccH** Device Address **AccTH:AccT** Memory Address
Output values	None
Destroy	None
Time	----
Observations	----

Example	Write a Byte 0x5a in device 5 at address 1000
	
	Ldi Acc,0x5a
	Ldi AccH,5
	Ldiawt 1000
	rcall _AT24C64_BYTE_WRITE
	call _AT24C64_BYTE_WRITE (chips >=16k)

Name	_AT24C64_BYTE_READ
Function	Read a byte from AT24C64 chip
Input values	**AccH** Device Address **AccTH:AccT** Memory Address
Output values	**Acc** Read Data
Destroy	None
Time	----
Observations	----
Example	Read a Byte in device 5 at address 1000 Ldi AccH,5 Ldiawt 1000 rcall _AT24C64_BYTE_READ call _AT24C64_BYTE_READ (chips >=16k)

AVRE2P (BE256.INC)

Description

Drive to write/read EEPROM in AVR device with EEPROM size <=256 bytes.

Implemented Functions

Name	**_EEPROM_WRITE**
Function	Write a byte into EEPROM
Input values	**Acc** Data to be write **AccH** Address
Output values	None
Destroy	None
Time	----
Observations	----

Example	Write a Byte 0x5a 5 at address 100
	
	Ldi **Acc**,0x5a
	Ldi **AccH**,100
	rcall _EEPROM_WRITE
	call _EEPROM_WRITE (chips >=16k)

Name	**_EEPROM_READ**
Function	Read a byte from EEPROM
Input values	**AccH** Address
Output values	**Acc** read Data
Destroy	None
Time	----
Observations	----
Example	Read a Byte at address 100 Ldi **AccH**,100 rcall _EEPROM_READ call _EEPROM_READ (chips >=16k)

AVRE2P (A256.INC)

Description

Drive to write/read EEPROM in AVR device with EEPROM size >256 bytes.

Implemented Functions

Name	**_EEPROM_WRITE**
Function	Write a byte into EEPROM
Input values	**Acc** Data to be write **AccT:AccH** Address
Output values	None
Destroy	None
Time	----
Observations	----

Example	Write a Byte 0x5a 5 at address 100
	
	Ldi **Acc**,0x5a
	Ldi **AccH**,low(100)
	Ldi **AccT**,high(100)
	rcall _EEPROM_WRITE
	call _EEPROM_WRITE (chips >=16k)

Name	**_EEPROM_READ**
Function	Read a byte from EEPROM
Input values	**AccT:AccH** Address
Output values	**Acc** read Data
Destroy	None
Time	----
Observations	----

Example	Read a Byte at address 100
	
	Ldi **AccH**,low(100)
	Ldi **AccT**,high(100)
	rcall _EEPROM_READ
	call _EEPROM_READ (chips >=16k)

DATA FLASH

AT45DB161-B

Description

The AT45DB161B is a 2.5-volt or 2.7-volt only, serial interface Flash memory ideally suited for a wide variety of digital voice-, image-, program code and data-storage applications. Its 17,301,504 bits of memory are organized as 4096 pages of 528 bytes each. In addition to the main memory, the AT45DB161B also contains two SRAM data buffers of 528 bytes each. The buffers allow receiving of data while a page in the main memory is being reprogrammed, as well as writing a continuous data stream. EEPROM emulation (bit or byte alterability) is easily handled with a self-contained three step Read-Modify-Write operation.Unlike conventional Flash memories that are accessed randomly with multiple address lines and a parallel interface, the DataFlash uses a SPI serial interface to sequentially access its data. DataFlash supports SPI mode 0 and mode 3. The simple serial interface facilitates hardware layout, increases system reliability, minimizes switching noise, and reduces package size and active pin count. The device is optimized for use in many commercial and industrial applications where high density, low pin count, low voltage, and low power are essential. The device operates at clock frequencies up to 20 MHz with a typical active read current consumption of 4 mA.

Implemented Functions

Name	_AT45DB161B_INIT
Function	Initialize AT45DB161B Interface
Input values	None
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disable during initialization.

Example	Initialize device using data port as PORTA, Reset Port as PORTB,WP port as PORTC, SO bit=0, SI bit=1, SCK bit=2,CS bit=3,WP bit=0,RESET bit=1
	
	`.EQU _AT45DB161B_PORT_OUTPUT = PORTA`
	`.EQU _AT45DB161B_PORT_DIR = DDRxA`
	`.EQU _AT45DB161B_PORT_INPUT = PINxA`
	
	`.EQU _AT45DB161B_RESET_OUTPUT= PORTB`
	`.EQU _AT45DB161B_RESET_DIR = DDRxB`
	`.EQU _AT45DB161B_RESET_INPUT = PINB`
	
	`.EQU _AT45DB161B_WP_OUTPUT = PORTC`
	`.EQU _AT45DB161B_WP_DIR = DDRC`
	`.EQU _AT45DB161B_WP_INPUT = PINC`
	
	`.EQU _AT45DB161B_SO_BIT = BIT0`
	`.EQU _AT45DB161B_SI_BIT = BIT1`
	`.EQU _AT45DB161B_SCK_BIT = BIT2`
	`.EQU _AT45DB161B_CS_BIT = BIT3`
	`.EQU _AT45DB161B_WP_BIT = BIT0`
	`.EQU _AT45DB161B_RESET_BIT = BIT1`
	
	`rcall _AT45DB161B_INIT`
	`call _AT45DB161B_INIT (chips >=16k)`

Name	_AT45DB161B_RESET
Function	Reset AT45DB161B
Input values	None
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disable during initialization.
Example	rcall _AT45DB161B_RESET call _AT45DB161B_RESET (chips >=16k)

Name	_AT45DB161B_DATA_OUT
Function	Send Data or Command AT45DB161B
Input values	**Acc** Data or Command
Output values	None
Destroy	None
Time	80 clocks
Observations	- Global interrupts are disable during initialization. SPI Mode 3 is asserted
Example	Send continuous_array_read command to chip ldi **Acc**,_AT45DB161B_SPI3_CMD_CONTINUOUS_ARRAY_READ rcall _AT45DB161B_DATA_OUT call _AT45DB161B_DATA_OUT (chips >=16k)

Name	_AT45DB161B_DATA_IN
Function	Read Data from AT45DB161B
Input values	None
Output values	**Acc** Data
Destroy	None
Time	44 clocks
Observations	- Global interrupts are disable during initialization. SPI Mode 3 is asserted
Example	rcall _AT45DB161B_DATA_IN call _AT45DB161B_DATA_IN (chips >=16k)

Name	_AT45DB161B_DATA_END
Function	Finish data transfer to AT45DB161B
Input values	None
Output values	None
Destroy	None
Time	---
Observations	- Global interrupts are disable during initialization. SPI Mode 3 is asserted
Example	rcall _AT45DB161B_DATA_END call _AT45DB161B_DATA_END (chips >=16k)

Name	_AT45DB161B_DATA_START
Function	Start data transfer to AT45DB161B
Input values	None
Output values	None
Destroy	None
Time	---
Observations	- Global interrupts are disable during initialization. SPI Mode 3 is asserted
Example	rcall _AT45DB161B_DATA_START call _AT45DB161B_DATA_START (chips >=16k)

Name	_AT45DB161B_SET_WRITE_PROTECT
Function	Set WP pin state of AT45DB161B
Input values	**Acc** _ON=Protect _OFF=release
Output values	None
Destroy	None
Time	---
Observations	---
Example	rcall _AT45DB161B_SET_WRITE_PROTECT call _AT45DB161B_SET_WRITE_PROTECT (chips >=16k)

Name	_AT45DB161B_GET_STATUS_REGISTER
Function	Get status register state of AT45DB161B
Input values	None
Output values	Acc status Bit 7=READY/BUSY state 1=READ 0=BUSY Bit 6=COMPARE 0=compare math memory Bit 5=1,bit 4=0,bit 3=1,bit 2=1, bit 1=x, bit 0=x
Destroy	None
Time	---
Observations	---
Example	rcall _AT45DB161B_GET_STATUS_REGISTER call _AT45DB161B_GET_STATUS_REGISTER (chips >=16k)

Name	_AT45DB161B_SET_ADDRESS
Function	Set start address of AT45DB161B
Input values	**X** Buffer Address **Y** Page Address
Output values	----
Destroy	None
Time	---
Observations	---

Example	Set page address = 1000 and buffer address = 50
	
	Ldiw **X**,50
	Ldiw **Y**,1000
	rcall _AT45DB161B_SET_ADDRESS
	call _AT45DB161B_SET_ADDRESS (chips >=16k)

Name	_AT45DB161B_CONTINUOUS_ARRAY_READ
Function	Send continuous array read command to AT45DB161B
Input values	X Buffer Address Y Page Address
Output values	----
Destroy	None
Time	---
Observations	---

Example	Send continuou read command to address = 1000 and buffer address = 50
	
	Ldiw X,50
	Ldiw Y,1000
	rcall _AT45DB161B_CONTINUOUS_ARRAY_READ
	call _AT45DB161B_CONTINUOUS_ARRAY_READ (chips >=16k)

Name	_AT45DB161B_BUFFER1_WRITE
Function	Send Buffer1 write to AT45DB161B
Input values	X Buffer Address
Output values	----
Destroy	None
Time	---
Observations	---
Example	Send Buffer1 write buffer address = 50 Ldiw X,50 rcall _AT45DB161B_BUFFER1_WRITE call _AT45DB161B_BUFFER1_WRITE (chips >=16k)

Name	_AT45DB161B_BUFFER1_READ
Function	Send Buffer1 Read to AT45DB161B
Input values	X Buffer Address
Output values	----
Destroy	None
Time	---
Observations	---
Example	Send Buffer1 READ buffer address = 50 Ldiw X,50 rcall _AT45DB161B_BUFFER1_READ call _AT45DB161B_BUFFER1_READ (chips >=16k)

Name	_AT45DB161B_BUFFER1_WRITE_INTO_PAGE
Function	program Buffer1 into Flash Page to AT45DB161B

Input values	**Y** Start Page Address
Output values	----
Destroy	None
Time	---
Observations	---
Example	Send program Buffer1 into Flash Page page = 50 Ldiw **Y**,50 rcall _AT45DB161B_BUFFER1_WRITE_INTO_PAGE call _AT45DB161B_BUFFER1_WRITE_INTO_PAGE (chips >=16k)

Name	**_AT45DB161B_BUFFER1_READ_FROM_PAGE**
Function	read Buffer1 from Flash Page to AT45DB161B
Input values	**Y** Start Page Address
Output values	----
Destroy	None
Time	---
Observations	---
Example	Send program Buffer1 into Flash Page page = 50 Ldiw **Y**,50 rcall _AT45DB161B_BUFFER1_READ_FROM_PAGE call _AT45DB161B_BUFFER1_READ_FROM_PAGE (chips >=16k)

AT45DB161B Constants	
Name	**Value**
_AT45DB161B_SPI3_CMD_CONTINUOUS_ARRAY_READ	0XE8
_AT45DB161B_SPI3_CMD_MAIN_MEMORY_PAGE_READ	0XD2
_AT45DB161B_SPI3_CMD_BUFFER1_READ	0XD4
_AT45DB161B_SPI3_CMD_BUFFER2_READ	0XD6
_AT45DB161B_SPI3_CMD_STATUS_REGISTER_READ	0XD7
_AT45DB161B_CMD_BUFFER1_WRITE	0X84
_AT45DB161B_CMD_BUFFER2_WRITE	0X87
_AT45DB161B_CMD_BUFFER1_PROGRAM_WITH_ERASE	0X83
_AT45DB161B_CMD_BUFFER1_PROGRAM_WITH_ERASE	0X84
_AT45DB161B_CMD_BUFFER1_PROGRAM_WITHOUT_ERASE	0X88
_AT45DB161B_CMD_BUFFER2_PROGRAM_WITHOUT_ERASE	0X89
_AT45DB161B_CMD_PAGE_ERASE	0X81
_AT45DB161B_CMD_BLOCK_ERASE	0X50
_AT45DB161B_CMD_PAGE_PROGRAM_THROUGH_BUFFER1	0X82
_AT45DB161B_CMD_PAGE_PROGRAM_THROUGH_BUFFER2	0X85
_AT45DB161B_CMD_MAIN_MEMORY_PAGE_TO_BUFFER1_TRANSFER	0X53
_AT45DB161B_CMD_MAIN_MEMORY_PAGE_TO_BUFFER2_TRANSFER	0X55
_AT45DB161B_CMD_MAIN_MEMORY_PAGE_TO_BUFFER1_COMPARE	0X60
_AT45DB161B_CMD_MAIN_MEMORY_PAGE_TO_BUFFER2_COMPARE	0X61
_AT45DB161B_CMD_AUTO_PAGE_REWRITE_THROUGH_BUFFER1	0X58
_AT45DB161B_CMD_AUTO_PAGE_REWRITE_THROUGH_BUFFER2	0X59
_AT45DB161B_RDY_BIT	7
_AT45DB161B_COMP_BIT	6

Analog to Digital Converters

ADC831

Description

ADC831 or TLC831 chip is a 8bit Analog to Digital Converter with serial control and diferencial input. This device is a 8bit successive approximation analog to digital converters. The serial output is configured to interface with standard shift registers or microprocessors.

Implemented Functions

Name	**ADC831_INIT**
Function	Initialize ADC831 Interface
Input values	None
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disable during initialization. - After this initialization CLK,CS,DATA are in high level, CLK,CS as output and Data as input.
Example	Define PORT where the ADC831 is connected in this example PORTB .EQU _ADC831_PORT_OUT= PORTB .EQU _ADC831_PORT_IN = PINB .EQU _ADC831_PORT_DIR= DDRB Define pin bit numbers .EQU _ADC831_BIT_CLK = 0 .EQU _ADC831_BIT_CS = 1 .EQU _ADC831_BIT_DATA= 2 Then initialize one of below two methods rcall _ADC831_INIT call _ADC831_INIT (chips >=16k)
Name	**_ADC831_GET**

Function	Get 8bit unsigned value from ADC831
Input values	None
Output values	**Acc** 8bit unsigned value
Destroy	Flags
Time	**Average conversion time** / **Frequency** 151.0uS / 1Mhz 37.6uS / 4Mhz 25.2uS / 6Mhz 18.9uS / 8Mhz 22.0uS / 10Mhz 20.1uS / 14.3Mhz 22.3uS / 16Mhz
Observations	---
Example	After below one of both calls **Acc** with ADC831 value rcall _ADC831_GET call _ADC831_GET (chips >=16k)

ADC832

Description

ADC832 or TLC832 chip is a 8bit Analog to Digital Converter with serial control and have 2 input channels. This device is a 8bit successive approximation analog to digital converters. The serial output is configured to interface with standard shift registers or microprocessors.

Implemented Functions

Name	**_ADC832_INIT**
Function	Initialize ADC832 Interface
Input values	None
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disable during initialization. - After this initialization CLK,CS,DATA_OUT,DATA_IN are in high level, CLK,CS,DATA_OUT as output and DATA_IN as input
Example	Define PORT where the ADC832 is connected in this example PORTB .EQU _ADC832_PORT_OUT= PORTB .EQU _ADC832_PORT_IN = PINB .EQU _ADC832_PORT_DIR= DDRB Define pin bit numbers .EQU _ADC832_BIT_CLK = 0 .EQU _ADC832_BIT_CS = 1 .EQU _ADC832_BIT_DATA_OUT = 2 .EQU _ADC832_BIT_DATA_IN=3 Then initialize one of below two methods rcall _ADC832_INIT call _ADC832_INIT (chips >=16k)

Name	_ADC832_GET
Function	Get 8bit unsigned value from ADC831
Input values	**Acc** Channel 0 or 1
Output values	**Acc** 8bit unsigned value
Destroy	Flags
Time	**Average conversion time** / **Frequency** 166.0uS / 1Mhz 41.5uS / 4Mhz 27.7uS / 6Mhz 20.7uS / 8Mhz 23.8uS / 10Mhz 21.7uS / 14.3Mhz 23.8uS / 16Mhz
Observations	---
Example	Set **Acc** with channel 1 after below one of both calls **Acc** with ADC832 value Ldi Acc,1 rcall _ADC831_GET call _ADC831_GET (chips >=16k)

ADC8535

Description

ADC8535 is a 10bit Analog to Digital Converter internal of AT90S8535 microcontroller or other equivalents of AVR family.

Implemented Functions

Name	**_ADC_INIT**
Function	Initialize internal ADC engine
Input values	**Acc** CLK prescaler factor in power of 2 1=2,2=4,3=8
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disable during initialization.
Example	Initialize ADC with prescaler = 2 Ldi **Acc**,2 rcall _ADC_INIT call _ADC_INIT (chips >= 16k)

Name	**_ADC_CHANNEL**
Function	Set multiplexed channel to be use
Input values	**Acc** channel number 0 to 7
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Set ADC channel to 3 Ldi **Acc**,3 rcall _ADC_CHANNEL call _ADC_CHANNEL (chips >=16k)

Name	**_ADC_GET**
Function	Get unsigned 10bit value from ADC
Input values	**None**
Output values	**AccH:Acc** has 10bit value
Destroy	Flags
Time	----
Observations	----
Example	After below both examples **AccH:Acc** have the 10bit value from ADC rcall _ADC_GET call _ADC_GET (chips >=16k)

ADCM128

Description

ADCM128 is a 10bit Analog to Digital Converter internal of ATMEGA128 microcontroller or other equivalents of AVR family.

Implemented Functions

Name	**_ADC_INIT**
Function	Initialize internal ADC engine
Input values	**Acc** CLK prescaler factor in power of 2 1=2,2=4,3=8
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disable during initialization.
Example	Initialize ADC with prescaler = 2 Ldi **Acc**,2 rcall _ADC_INIT call _ADC_INIT (chips >=16k)

Name	**_ADC_SET_CHANNEL**
Function	Set multiplexed channel to be use
Input values	**Acc** channel number 0 to 7
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	Set ADC channel to 3 Ldi **Acc**,3 rcall _ADC_CHANNEL call _ADC_CHANNEL (chips >=16k)

Name	_ADC_GET_CHANNEL
Function	Get multiplexed channel in use
Input values	None
Output values	**Acc** channel number 0 to 7
Destroy	None
Time	----
Observations	----
Example	Get ADC channel in use, After below of both calls **Acc** has the channel number in use. rcall _ADC_GET_CHANNEL call _ADC_GET_CHANNEL (For chips with more 16k)

Name	_ADC_GET_VALUE
Function	Get unsigned 10bit value from ADC
Input values	**None**
Output values	**AccH:Acc** has 10bit value
Destroy	Flags
Time	----
Observations	----
Example	After below both examples **AccH:Acc** have the 10bit value from ADC rcall _ADC_GET_VALUE call _ADC_GET_VALUE(chips >=16k)

ADS8320(Texas Instruments/Burr-Brown)

Description

The ADS8320 is a 16-bit sampling analog-to-digital converter with guaranteed specifications over a 2.7V to 5.25V supply range. It requires very little power even when operating at the full 100kHz data rate. At lower data rates, the high speed of the device enables it to spend most of its time in the power-down mode the average power dissipation is less than 100mW at 10kHz data rate

Implemented Functions

Name	**_ADS8320_INIT**
Function	Initialize ADS8320 interface
Input values	None
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disable during initialization.
Example	Define PORT where the ADS8320 is connected in this example PORTB .EQU _ADS8320_PORT_OUT= PORTB .EQU _ADS8320_PORT_IN = PINB .EQU _ADS8320_PORT_DIR= DDRB Define pin bit numbers .EQU _ADS832_BIT_CLK = 0 .EQU _ADS832_BIT_CS = 1 .EQU _ADS832_BIT_DATA= 2 Then initialize one of below two methods rcall _ADS8320_INIT call _ADS8320_INIT (Chips >=16k)

Name	_ADS8320_GET
Function	Get 16bit value from ADS8320
Input values	None
Output values	**AccH:Acc** 16bit value
Destroy	Flags
Time	----
Observations	----
Example	After below both examples **AccH:Acc** have the 16bit value from ADC rcall _ADS8320_GET_VALUE call _ADS8320_GET_VALUE(chips >=16k)

Astronomy

Julian Day

Description

The Julian day or Julian day number (JDN) is the integer number of days that have elapsed since an initial epoch defined as noon Universal Time (UT) Monday, January 1, 4713 BC in the proleptic Julian calendar. That noon-to-noon day is counted as Julian day 0. Negative values can also be used, although those predate all recorded history. Now, at 18:55, Thursday August 28, 2008 (UTC) the JDN is 2454707.

The Julian date (JD) is a continuous count of days and fractions elapsed since the same initial epoch. Currently the JD is 2454707.28844. The integral part (its floor) gives the Julian day number. The fractional part gives the time of day since noon UT as a decimal fraction of one day or fractional day, with 0.5 representing midnight UT. Typically, a 64-bit floating point (double precision) variable can represent an epoch expressed as a Julian date to about 1 millisecond precision. This routine compute Julian day according below equation:

$$y = year$$

$$m = month$$

$$d = day$$

If m<=2 then

$$m = m + 12$$

$$y = y - 1$$

If calendar is Julian then,

$$b = -2 + fix\left(\frac{y + 4716}{4}\right) - 1179$$

Else

$$b = fix\left(\frac{y}{400}\right) - fix\left(\frac{y}{100}\right) + fix\left(\frac{y}{4}\right)$$

Then

$$a = 365 * y + 1720996.5$$

And

$$Julian\ day = a + b + fix\big(30.6001 * (m + 1)\big) + d$$

Implemented Functions

Name	**_JULIAN_DAY_INIT**
Function	Initialize _JULIAN_DAY Object
Input values	None
Output values	None
Destroy	Flags, register R0..R15
Time	----
Observations	- Global interrupts are disable during initialization.
Example	Call below one of both methods to initialize rcall _JULIAN_DAY_INIT call _JULIAN_DAY_INIT (Chips >= 16k)

Name	_JULIAN_DAY_SET_DATE
Function	Set desiderated date for computation
Input values	**Acc** Day 1..31 **AccH** Month 1..12 **AccTH:AccT** Year
Output values	None
Destroy	None
Time	----
Observations	----

Example	Set date to Abril 6 of 1964
	
	Ldi **Acc**,6
	Ldi **AccH**,4
	Ldiawt 1964
	rcall _JULIAN_DAY_SET_INIT
	call _JULIAN_DAY_SET_INIT (chips >= 16k)

Name	_JULIAN_DAY_GET_DATE
Function	Get date
Input values	None
Output values	**Acc** Day 1..31 **AccH** Month 1..12 **AccTH:AccT** Year
Destroy	None
Time	----
Observations	----
Example	After below one both calls **Acc**=Day, **AccH**=Month, **AccTH:AccT**=Year rcall _JULIAN_DAY_GET_INIT call _JULIAN_DAY_GET_INIT (chips >=16k)

Name	_JULIAN_DAY_SET_GREGORIAN
Function	Set Gregorian date
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	After below one both calls date is assume Gregorian rcall _JULIAN_DAY_SET_GREGORIAN call _JULIAN_DAY_GET_GREGORIAN (chips >= 16k)

Name	_JULIAN_DAY_SET_JULIAN
Function	Set Julian date
Input values	None
Output values	None
Destroy	None
Time	----
Observations	----
Example	After below one both calls date is assume Julian rcall _JULIAN_DAY_SET_JULIAN call _JULIAN_DAY_GET_JULIAN (chips >= 16k)

Name	_JULIAN_DAY_GET_VALUE
Function	Get a pointer of Julian day value in Float Double precision
Input values	None
Output values	Z → Fload Double Value
Destroy	None
Time	----
Observations	The Julian day returned for this function is updated only before calling _JULIAN_DAY_COMPUTE function
Example	After below one both calls date is Z point to Julian day Float Double Value rcall _JULIAN_DAY_GET_VALUE call _JULIAN_DAY_GET_VALUE (chips >= 16k)

Name	_JULIAN_DAY_COMPUTE
Function	Using actual date compute Julian day
Input values	None
Output values	None
Destroy	Flags, R0..R15
Time	----
Observations	if calling this routine before setting a date and gregorian flag a unpredictable result will be returned
Example	After below one both calls Julian Day has computed. rcall _JULIAN_DAY_COMPUTE call _JULIAN_DAY_COMPUTE (chips >= 16k)

COMM - Communications

I2C (I2C.INC)

Description

I2C (Inter-Integrated Circuit) is a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone. The name is pronounced *eye-squared-see* or *eye-two-see*. As of October 1, 2006, no licensing fees are required to implement the I2C protocol. However, fees are still required in order to "officially" allocate I2C slave addresses. This Author drive implement function to implement I2C at any (no mapped) port.

Implemented Functions

Name	**_I2C_INIT**
Function	Initialize I2C Interface
Input values	None
Output values	None
Destroy	----
Time	----
Observations	- Global interrupts are disable during initialization.
Example	Initialize I2C at PORTB with SCL=0 and SDA=1 .EQU _I2C_PORT_OUT = PORTB .EQU _I2C_PORT_IN = PINB .EQU _I2C_PORT_DIR = DDRB .EQU _I2C_BIT_SCL = 0 .EQU _I2C_BIT_SDA = 1 rcall _I2C_INIT call _I2C_INIT (Chips >= 16k)

Name	_I2C_START
Function	Insert I2C start conditions
Input values	None
Output values	None
Destroy	----
Time	----
Observations	----
Example	rcall _I2C_START call _I2C_START (Chips >= 16k)

Name	_I2C_STOP
Function	Insert I2C start conditions
Input values	None
Output values	None
Destroy	----
Time	----
Observations	----
Example	rcall _I2C_START call _I2C_START (Chips >= 16k)

Name	_I2C_BIT_OUT
Function	Send Bit thru I2C line
Input values	**Cy** bit to be send cy=1 to ONE
Output values	None
Destroy	----
Time	----
Observations	Usually this routine is used only by _I2C_BYTE_OUT routine, be carefully when use directly
Example	Send bit 1 sec rcall _I2C_BIT_OUT call _I2C_BIT_OUT (Chips >= 16k)

Name	_I2C_BIT_IN
Function	Get Bit from I2C line
Input values	None
Output values	**Cy** bit read
Destroy	----
Time	----
Observations	Usually this routine is used only by _I2C_BYTE_IN routine, be carefully when use directly
Example	rcall _I2C_BIT_IN call _I2C_BIT_IN (Chips >= 16k)

Name	_I2C_BYTE_OUT
Function	Send a byte thru I2C line
Input values	**Acc** Data to be send
Output values	None
Destroy	----
Time	----
Observations	----
Example	Send data 0xaa to I2C line Ldi **Acc**,0xaa rcall _I2C_DATA_OUT call _I2C_DATA_OUT (Chips >= 16k)

Name	_I2C_BYTE_IN
Function	Get a byte from I2C line
Input values	None
Output values	**Acc** Read Data
Destroy	----
Time	----
Observations	----
Example	rcall _I2C_DATA_IN call _I2C_DATA_IN (Chips >= 16k)

Name	**_I2C_ACK_IN**
Function	Get a Ack=acknowledgement from I2C line
Input values	None
Output values	**Cy**
Destroy	----
Time	----
Observations	----
Example	rcall _I2C_ACK_IN call _I2C_ACK_IN (Chips >= 16k)

N64 (N64_COMM.INC)

Description

This drive implements N64 Joystick controller of Nintendo corp. Implements routines allow user get all button and Analog Joystick values and states.

Implemented Functions

Name	**_N64_INIT**
Function	Initialize _N64 Interface
Input values	None
Output values	None
Destroy	----
Time	----
Observations	- Global interrupts are disable during initialization. - After this initialization DATAINOUT is configured as input to prevent short circuit during initialization.
Example	Initialize N64 controller at PORTB and Data bit=0 .EQU _N64_DATAINOUT_OUT = PORTB .EQU _N64_DATAINOUT_IN = PINB .EQU _N64_DATAINOUT_DIR = DDRB .EQU _N64_BIT_DATAINOUT = 0 rcall _N64_INIT call _N64_INIT (Chips >= 16k)

Name	_N64_STATUS
Function	Update button and joystick coordinates of _N64
Input values	None
Output values	None
Destroy	Flags
Time	----
Observations	----
Example	rcall _N64_STATUS call _N64_STATUS (Chips >= 16k)

Name	_N64_GET_A
Function	Get key states
Input values	Acc Key Code
Output values	Acc _ON=pressed or _OFF=relesed
Destroy	Flags
Time	----
Observations	----
Example	Read status of button Z Ldi Acc,_N64_KEY_A rcall _N64_GET_A call _N64_GET_A (Chips >= 16k)

Name	_N64_GET_X
Function	Get joystick X coordinate
Input values	None
Output values	Acc X coordinate
Destroy	Flags
Time	----
Observations	----
Example	rcall _N64_GET_X call _N64_GET_X (Chips >= 16k)

Name	_N64_GET_Y
Function	Get joystick Y coordinate
Input values	None
Output values	Acc Y coordinate
Destroy	Flags
Time	----
Observations	----
Example	rcall _N64_GET_Y call _N64_GET_Y (Chips >= 16k)

SLIP (SLIP.INC)

Description

The TCP/IP protocol family runs over a variety of network media: IEEE 802.3 (ethernet) and 802.5 (token ring) LAN's, X.25 lines, satellite links, and serial lines. There are standard encapsulations for IP packets defined for many of these networks, but there is no standard for serial lines. SLIP, Serial Line IP, is a currently a de facto standard, commonly used for point-to-point serial connections running TCP/IP. It is not an Internet standard. More details see RFC 1055.

SLIP EXAMPLE

Fortunately, one of the TCP/IP families of standards,SLIP, provides exactly this functionality. It uses simple escape codes inserted in the serial data stream to signal block boundaries as follows.

- The end of each block is signaled by a special End byte, with a falue of 0xC0.
- If a data byte equal 0XC0, two bytes with the values 0xDB,0XDC are sent instead.
- if a data byte equal 0xDBH,two bytes with the values 0xDB,0xDD are sent instead.

Additionally, most implementation send the End byte at the beginning of each block to clear out garbage characters prior to starting the new message.

SLIP FRAME

END 0XC0	DATA 1-1006 BYTES	END 0XC0

|

There is effectively no limit to the size of the data block, but you have to decide on some value in order to dimension the data bufers. With old, slow serial links, a maximum size of 256 bytes was generally used, but you'll be using faster links, and a larger size is better for minimizeng protocol overhead. By convention, 1006 bytes is oten used.

The encoding method can best be illustrated by an example. Assume a six-byte block of data with the hex values BF C0 C1 DB DC is sent; it is expanded to C0 BF DB DC C1 DB DD DC C0.

SLIP TRANSMISSION SAMPLE

```
                  +----+----+----+----+----+
block data flow   |0XBF|0XC0|0XC1|0XDB|0XDC|
                  +----+----+----+----+----+
                    |     |    |    |    |
                    |     |    |    |    |
                    |     |    |    |    +--------------+
                    |     |    |    +-----------+       |
                    |     |    +--------+       |       |
                    |     +------+      |       |       |
                    |            |      |       |       |
                    +-----+      |      |       |       |
                          |   +--+--+   |    +--+--+    |
                          V   V     V   V    V     V    V
                  +----+----+----+----+----+----+----+----+----+
 SLIP coded flow  |0XC0|0XBF|0XDB|0XDC|0XC1|0XDB|0XDD|0XDC|0XCO|
                  +----+----+----+----+----+----+----+----+----+
                  END                                     END
```

SLIP OBSERVATION

When connect SLIP device in serial port under WINDOWS, it send a ATE1<cr> modem command and device replay OK<cr><lf>, after this the OS send DSVP packed to inform device about OS resorces, device replay OK<cr><lf> again, after this OS send a SLIP protocol with TCP/IP.

Connections

MODEM for SLIP protocol

```
     +--------+                     +--------+
     | MODEM |                      |  PC    |
     |       RTS<------------------>CTS      |
     |       CTS<------------------>RTS      |
     |       DCD<------------------>DCD      |
     |       TXD<------------------>RXD      |
     |       RXD<------------------>TXD      |
     |       |                      |        |
     +--------+                     +--------+
```

Generic serial DEVICE for SLIP protocol

```
     +--------+                     +--------+
     | MODEM |                      |  PC    |
     |       |                  +-->CTS      |
     |       |                  +-->RTS      |
     |       |                  +-->DCD      |
     |       TXD<-------------------->RXD    |
     |       RXD<-------------------->TXD    |
     |       |                      |        |
     +--------+                     +--------+
```

CTS,RTS,DCD from PC side are connected together

Implemented Functions

Sample code of **SLIP** Initialization sequence.

```
ldiaw      RX_BUFFER_SIZE          ;rx buffer size
ldiw       Z,RX_BUFFER_PTR         ;rx pointer
RCALL      _SLIP_SET_RX_BUFFER     ;set
ldiaw      TX_BUFFER_SIZE          ;tx buffer size
ldiw       Z,RX_BUFFER_PTR         ;tx pointer
RCALL      _SLIP_SET_TX_BUFFER     ;set
ldiw       Z,RX_FUNC_ADDR          ;set function rx address
RCALL      _SLIP_SET_RX_ADDR
ldiw       Z,TX_FUNC_ADDR          ;set function tx address
RCALL      SLIP_SET_TX_ADDR
ldiw       Z,TIMEOUT_FUNC_ADDR     ;set timeout function
ldiaw      100                     ;set timeout to 100ms
RCALL      _SLIP_SET_TIMEOUT_ADDR
```

Name	**_SLIP_SET_RX_BUFFER**
Function	Set address of receiver data buffer and size
Input values	**AccH:Acc** size **Z->** data buffer
Output values	None
Destroy	----
Time	----
Observations	----
Example	Set SLIP receiver data buffer address and size Ldiaw SLIP_SIZE Ldiw Z,SLIP_BUFFER rcall _SLIP_SET_RX_BUFFER call _SLIP_SET_RX_BUFFER(Chips >= 16k)

Name	**_SLIP_GET_RX_BUFFER**
Function	Get address of receiver data buffer and size
Input values	None
Output values	**AccH:Acc** size **Z->** data buffer
Destroy	----
Time	----
Observations	----
Example	rcall _SLIP_GET_RX_BUFFER call _SLIP_GET_RX_BUFFER(Chips >= 16k)

Name	_SLIP_SET_TX_BUFFER
Function	Set address of transmitter data buffer and size
Input values	AccH:Acc size Z-> data buffer
Output values	None
Destroy	----
Time	----
Observations	----
Example	Set SLIP transmitter data buffer address and size Ldiaw SLIP_SIZE Ldiw Z,SLIP_BUFFER rcall _SLIP_SET_TX_BUFFER call _SLIP_SET_TX_BUFFER(Chips >= 16k)

Name	_SLIP_GET_TX_BUFFER
Function	Get address of transmitter data buffer and size
Input values	None
Output values	AccH:Acc size Z-> data buffer
Destroy	----
Time	----
Observations	----
Example	rcall _SLIP_GET_TX_BUFFER call _SLIP_GET_TX_BUFFER(Chips >= 16k)

Name	_SLIP_SET_RX_ADDRESS
Function	Set address of routine that receiver data
Input values	Z address
Output values	None
Destroy	----
Time	----
Observations	----
Example	Set SLIP receiver address to RX_FUNCTION Ldiw Z,RX_FUNCTION rcall _SLIP_SET_RX_ADDRESS call _SLIP_SET_RX_ADDRESS(Chips >= 16k)

Name	_SLIP_SET_TX_ADDRESS
Function	Set address of routine that transmit data
Input values	**Z** address
Output values	None
Destroy	----
Time	----
Observations	----
Example	Set SLIP transmit address to TX_FUNCTION Ldiw Z,TX_FUNCTION rcall _SLIP_SET_TX_ADDRESS call _SLIP_SET_TX_ADDRESS(Chips >= 16k)

Name	_SLIP_SET_TIMEOUT_ADDRESS
Function	Set address of routine that set timeout value for received data
Input values	**Z** address
Output values	None
Destroy	----
Time	----
Observations	----
Example	Set SLIP timeout receiver address to TIMOUT_FUNCTION Ldiw Z,TIMOUT_FUNCTION rcall _SLIP_SET_TIMEOUT_ADDRESS call _SLIP_SET_TIMEOUT_ADDRESS(Chips >= 16k)

Name	_SLIP_SET_INDEX
Function	Set Data buffer index
Input values	**AccH:Acc** data index **Cy=1** if data>SLIP Buffer size
Output values	None
Destroy	----
Time	----
Observations	----
Example	Set SLIP index to get 3rd data Ldiaw 2 rcall _SLIP_SET_INDEX call _SLIP_SET_INDEX(Chips >= 16k)

Name	_SLIP_GET_INDEX
Function	Get Data buffer index
Input values	None
Output values	AccH:Acc data index
Destroy	----
Time	----
Observations	----
Example	rcall _SLIP_GET_INDEX call _SLIP_GET_INDEX(Chips >= 16k)

Name	_SLIP_GET_DATA
Function	Get Data from receiver buffer
Input values	None
Output values	Acc data
Destroy	----
Time	----
Observations	----
Example	rcall _SLIP_GET_DATA call _SLIP_GET_DATA(Chips >= 16k)

Name	_SLIP_POLLING
Function	Polling a serial line and waiting CODE_END
Input values	None
Output values	Acc=SLIP_MSG_OK,cy=0 if SLIP block received ok Acc=SLIP_MSG_POL_END,cy=1 SLIP packet not received Acc=SLIP_MSG_TIMEOUT,cy=1 if timeout occur Acc=SLIP_MSG_UNEXPECTED,expected ESC_END,but received other code AccTH:AccT total received bytes if _OK AccTH:AccT total received bytes until Error if _NOTOK
Destroy	----
Time	----
Observations	----

Name	_SLIP_SEND
Function	Transmitting Data using SLIP protocol
Input values	None
Output values	None
Destroy	----
Time	----
Observations	call _SLIP_SET_TX_BUFFER before to set address of data
Example	rcall _SLIP_SEND call _SLIP_SEND(Chips >= 16k)

_SLIP Constants	
Name	**Value**
_SLIP_CODE_END	0XC0
_SLIP_CODE_ESC	0XDB
_SLIP_CODE_ESC_END	0XDC
_SLIP_CODE_ESC_ESC	0XDD
_SLIP_CODE_OK	1
_SLIP_CODE_POL_END	2
_SLIP_CODE_TIMEOUT	3
_SLIP_CODE_UNEXPECTED	4

SERIAL

TWO_WIRE

DT_COMM_V1 (DT_COMM_V1.INC)

Description

DTCOMMV1 is a Author proprietary protocol that allow faster communication unilateral from Master to slave using only one wire where master always request transmission and slave return data according below flow chart, maximum rate obtained is about 125Kbits.

```
         Tr         bt
DATA ---+   +-------+  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
        |   |       |  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |10|11|12|13|14|15|P |1 |
        +---+       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
            ...RT...
        ^
        |
        Master Request
Tr = 4us
Bt = 8us
Rt = Slave answer time
0..15 data bits
P  = parity bit
1  = stop bit
```

Name	**_DTCOMMV1_SLAVE_INIT**
Function	Initialize DTCOMMV1 like a Slave interface
Input values	**None**
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disabled - Use External 0 interrupt handle _HDC_INT0_VECT
Example	rcall _DTCOMMV1_SLAVE_INIT call _DTCOMMV1_SLAVE_INIT(Chips >= 16k)

Name	_DTCOMMV1_MASTER_INIT
Function	Initialize DTCOMMV1 like a Master interface
Input values	**None**
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disabled
Example	Initialize master in PORTD Data bit=2 .EQU _DTCOMMV1_PORT_OUTPUT = PORTD .EQU _DTCOMMV1_PORT_DIR = DDRD .EQU _DTCOMMV1_PORT_INPUT = PIND .EQU _DTCOMMV1_DATA_BIT = 2 rcall _DTCOMMV1_SLAVE_INIT call _DTCOMMV1_SLAVE_INIT(Chips >= 16k)

Name	_DTCOMMV1_GET_DATA
Function	Get data from Slave
Input values	**None**
Output values	**AccH:Acc** Data
Destroy	None
Time	----
Observations	Call _DTCOMMV1_REQUEST Before to check if new data arrived
Example	rcall _DTCOMMV1_GET_DATA call _DTCOMMV1_GET_DATA(Chips >= 16k)

SOFTWARE (SOFTWARE SERIAL.INC)

Description

Software serial communication that allow use any port (not mapped) in any bit, this version work only 8 data bits 115200 bauds and 2 stop bits optimized to use 16Mhz crystal.

Name	**_SCOMM_INIT**
Function	Initialize SCOMM interface
Input values	None
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disabled during initialization

Example	Initialize SCOMM ports with Data Port as PORTA and control port as PORTC, TX bit=0, RX bit=1, RTS bit=2 CTS bit=3
	
	`.EQU _SCOMM_PORT_DATA_OUTPUT=PORTA`
	`.EQU _SCOMM_PORT_DATA_DIR    =DDRA`
	`.EQU _SCOMM_PORT_DATA_INPUT =PINA`
	
	`.EQU _SCOMM_PORT_CTRL_OUTPUT=PORTC`
	`.EQU _SCOMM_PORT_CTRL_DIR    =DDRC`
	`.EQU _SCOMM_PORT_CTRL_INPUT =PINC`
	
	`.EQU _SCOMM_TX_BIT  = 0`
	`.EQU _SCOMM_RX_BIT  = 1`
	`.EQU _SCOMM_RTS_BIT = 2`
	`.EQU _SCOMM_CTS_BIT = 3`
	
	`rcall _SCOMM_INIT`
	`call  _SCOMM_INIT(Chips >= 16k)`

Name	_SCOMM_TX
Function	Send data to Serial Line
Input values	Acc Data to be send
Output values	None
Destroy	None
Time	----
Observations	- Global interrupts are disabled during Transmission
Example	Send 0x27 to serial line Ldi **Acc**,0x27 rcall _SCOMM_TX call _SCOMM_TX(Chips >= 16k)

Name	_SCOMM_RX
Function	Get data to Serial Line
Input values	None
Output values	**Acc** Data received if cy=0 **Acc** Error Code if cy=1
Destroy	None
Time	----
Observations	- This routine use a fixed timeout of approximately 0.5 seconds elapsed this time **cy**=1 means timeout error
Example	rcall _SCOMM_RX call _SCOMM_RX(Chips >= 16k)

Name	_SCOMM_GET_RTS
Function	Get RTS state
Input values	None
Output values	**Cy=1** if RTS=high level
Destroy	None
Time	----
Observations	----
Example	rcall _SCOMM_GET_RTS call _SCOMM_GET_RTS(Chips >= 16k)

Name	_SCOMM_SET_CTS
Function	Set CTS state
Input values	CY=1 if to put CTS=HIGH
Output values	None
Destroy	None
Time	----
Observations	----
Example	Set CTS=low clc rcall _SCOMM_SET_CTS call _SCOMM_SET_CTS(Chips >= 16k)

REPRODUCTION AND COPYRIGHT.

Contact

dartagnanmath@gmail.com