

Université d'Ottawa  
Faculté de génie

École d'ingénierie et de  
technologies de  
l'information



University of Ottawa  
Faculty of Engineering

School of Information  
Technology and  
Engineering

**COURSE:** CEG3131/CEG3531

**SEMESTER:** Fall 2007

**PROFESSORS:** Voicu Groza  
Gilbert Arbez  
**DATE:** Dec 12, 2007  
**TIME:** 9h30 – 12h30

**Computer Architecture II**  
**FINAL EXAMINATION**

**NAME and STUDENT NUMBER:** \_\_\_\_\_ / \_\_\_\_\_

**Instructions:**

- Answer ALL questions on the examination paper.
- This is a close-book examination.
- Use the provided space to answer the following questions. If more space is needed, use the back of the page.
- Show all your calculations to obtain full marks.
- Calculators are allowed.
- Read all the questions carefully before you start.
- Pages starting at page 10 can be detached from the exam.  
(total pages 19)

1. There are three (3) questions in this examination.

<b>Question 1</b>	<b>SCI Module</b>	<b>20 marks</b>	
<b>Question 2</b>	<b>ATD Module</b>	<b>25 marks</b>	
<b>Question 3</b>	<b>Timer Module</b>	<b>35 marks</b>	
<b>Total</b>		<b>80 marks</b>	



**Question 2 Design - Analog to Digital Conversion (total 25 points)****QUESTION 2.1 ATD Quantization (10 points)**

Describe the Quantization method used by the HCS12 ATD. Include a diagram and algorithm flowchart to aid your description.

**QUESTION 2.2 Analog to Digital Conversion. (15 points)**

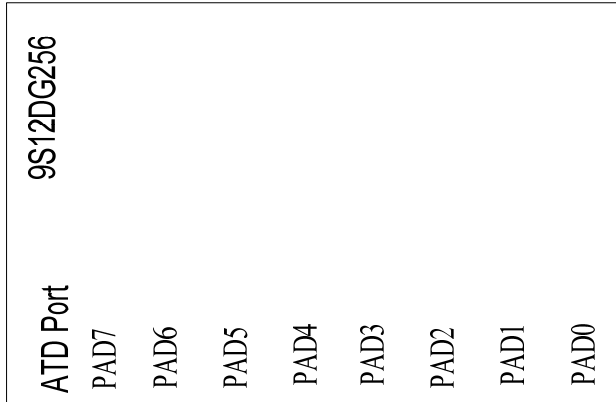
A signal with the following characteristics needs to be monitored by the HCS12 MCU:

- The signal contains frequencies between 10,000 and 40,000 Hertz.
- The signal varies between 0 and 200 mV.

The HCS12 MCU has the following pin connected as shown:

- $V_{SSA} = V_{LREF} = 0V$
- $V_{DDA} = V_{HREF} = 5V$

- a) (5 points) Complete the following diagram to show how the signal is conditioned before connecting it to an ATD pin. You must add a circuit (include specific component values).



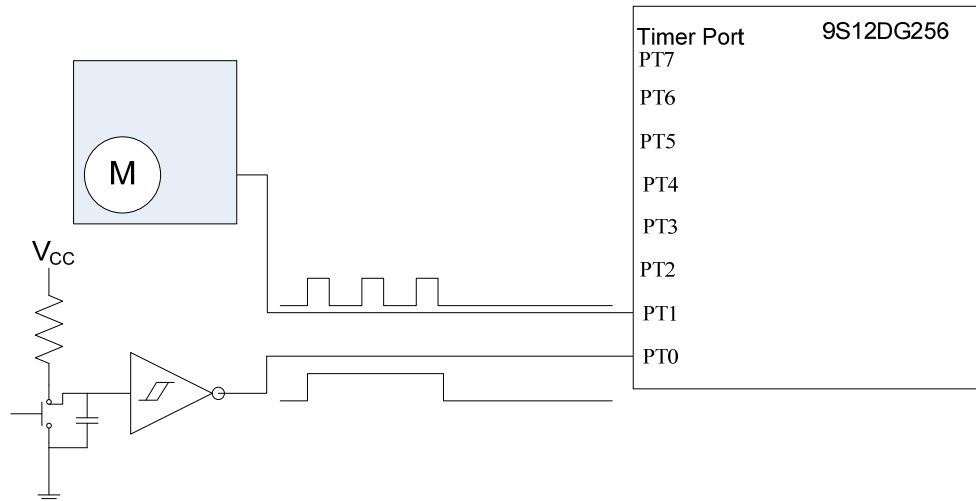
Analog Signal  
(10000-40000Hz)  
(0 to 200 mV)

- b) **(10 points)** The signal has to be continuously monitored. Assume that low sample-and-hold time is possible and 8bit quantization is suitable. Continuous sampling is required and interrupts shall be used to read conversion data into a buffer. The bus clock is set to 24MHz.
- How many ATD clocks are needed to perform a conversion?
  - What is the minimum sampling frequency for the given signal?
  - Find the period of the ATD clock (as an integer number of  $\mu\text{s}$ ) that complies with the above two questions.
  - Describe how to initialise the control registers of the HCS12 ATD.

To receive partial marks, do provide steps and/or a description on how to arrive at your settings.

### Question 3 – Timer (35 points)

Consider the following circuit where the HCS12 Timer provides the control of a motor M driven by a digital signal. When the push button switch is pushed, the motor M is turned on. When released, the motor is turned off.



The signal used to drive the motor circuit has a frequency of 1000 Hz with a duty cycle of 40% (that is the signal is high during 40% of a single cycle and low during 60% of a single cycle).

#### QUESTION 3.1 General (5 points)

a) Would you configure the Timer channel 0 for input-capture or output-compare? List the events (capture events if configured as an input-capture channel or the compare/output events if configured as an output-compare channel) of interest for this application for the channel.

b) Would you configure the Timer channel 1 for input-capture or output-compare? List the events (capture events if configured as an input-capture channel or the compare/output events if configured as an output-compare channel) of interest for this application for the channel.

**QUESTION 3.2 Initialization (5 points)**

Provide an initialization routine for the Timer module. In your design you shall be using interrupts with Timer Channel 1 (to generate the signal that drives the motor circuit). Note that interrupts must not be enabled by this routine.

```
; Subroutine: initTimer  
; Description
```

```
initTimer:
```

**QUESTION 3.3 Monitoring the Push Button (5 points)**

Provide the *monitorPushButton* routine that monitors the push button. It shall call the *motorOn* subprogram and *motorOff* subprograms when the appropriate event occurs in this channel. Note that this is NOT the main (i.e. you do not need to be concerned with any initialization). Assumed that the subroutine will be called after all initialization is complete.

```
; Subroutine: monitorPushButton  
; Description
```

```
monitorPushButton:
```



**QUESTION 3.4 Controlling the motor (20 points total)**

a) (**10 points**) Provide the subroutines `motorOn` and `motorOff` to control the generation of the signal in Timer Channel 1.

```
; Subroutine: motorOn  
; Description
```

`motorOn`

```
; Subroutine: motorOff  
; Description
```

`motorOff`

b) (**10 points**) Provide the ISR for the timer that will generate the signal on the pin of the timer channel 1.

```
; ISR: tc1_isr  
; Description
```

```
tc1_isr
```

## Annex A

### Transmission Module using a Circular Buffer

A Transmission Module has been developed to use a circular buffer for transmission of characters. Three variables are used to manipulate a circular character buffer (at address *buffer* with length *BUFLen*). The *bufStart* and *bufEnd* variables are pointers (contain addresses) to the start and end respectively of a string of characters to be transmitted. The variable *empty* indicates that the buffer is empty when set to TRUE (*empty* can only be set to TRUE when *bufStart* equals *bufEnd*). The software module contains the following subroutines and ISR:

- o **initTx**: Initialisation of the module consists of initialising the buffer variables and the SCI0.
- o **addChBuf**: Add a character to the buffer. The subroutine waits when the buffer is full (by comparing *bufStart* and *bufEnd* for equality and that *empty* is FALSE). Also the transmission interrupt is disabled while manipulating the buffer global variables and re-enable afterwards.
- o **getChBuf**: Removes a character from the buffer. It is called by the ISR for the transmission of a character.
- o **incPointer**: The subroutine is used to increment pointers to the circular buffer. When the pointer reaches the end of the buffer (*buffer*+*BUFLen*), it is reset to *buffer*. This subroutine is used to increment the values of both the *bufStart* and *bufEnd*.
- o **sco\_isr**: This ISR services interrupts from the SCI0 module. It obtains a character from the buffer using *getChBuf* and stores it in the SCI data register for transmission. If the buffer is empty it turns off the transmission interrupt.

The following provides the assembled code for the transmit module subroutines and ISR. The initialisation routine *initTx* is not provided.

```

74: ;-----
75: ;Subroutine: addChrBuf(chr)
76: ; Parameter: chr - in acc A - character to buffer
77: ; Global variables
78: ;     empty - set to TRUE when buffer is empty
79: ;     bufEnd - points to the end of the buffer (next place to write char)
80: ; Description:
81: ;     Adds character to the end of the buffer. Waits if buffer is full
82: ;     for transmit module to release space. SCI interrupt is disabled
83: ;     when writing to the buffer and then re-enabled. Re-enabled can
84: ;     also serve to start the transmission when transmit module is not
85: ;     processing characters (i.e. when writing into an empty buffer).
86: ;     Transmit LED turn on when re-enabling interrupts
87: ;
88:      212F          addChBuf:
89:      212F 34          pshx      ; preserve
90:      2130 FE 2502    aCBlp     ldx  bufEnd
91:      2133 BE 2500          cpx  bufStart
92:      2136 26 05          bne  aCont
93:      2138 F7 2504          tst  empty          ; is BUFLen TRUE
94:      213B 26 F3          bne  aCBlp
95:                      ; space available - lets add to buffer
96:      213D 4D CB 80    aCont     bclr SC0CR2,%10000000    ; Disable the interrupt
97:      2140 6A 00          staa  0,x          ; [bufEnd] <- chr
98:      2142 180B 00 2504    movb  #FALSE,empty          ; set empty to FALSE
99:      2147 16 2177          jsr  incPointer          ; increments the pointer in X
100:     214A 7E 2502          stx  bufEnd
101:     214D 4C CB 80    setInt     bset SC0CR2,%10000000    ; Enable the interrupt
102:     2150 30          pulx  ; restore
103:     2151 3D          rts
104:

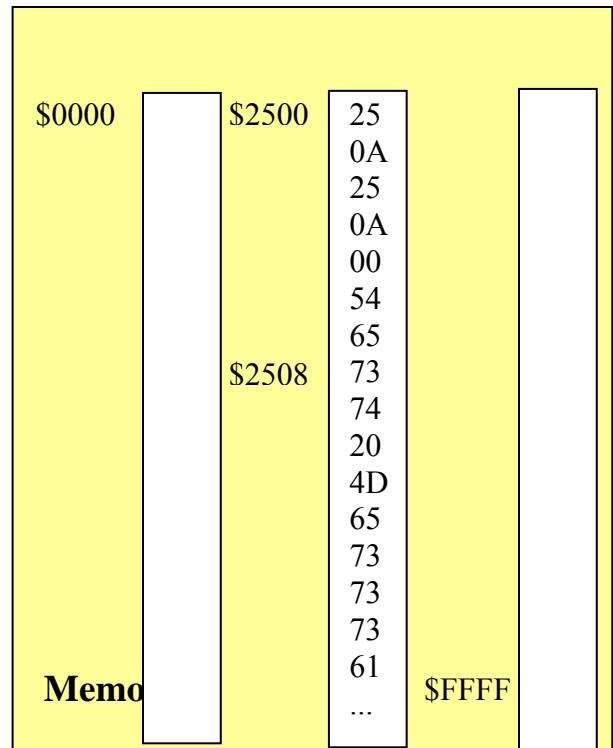
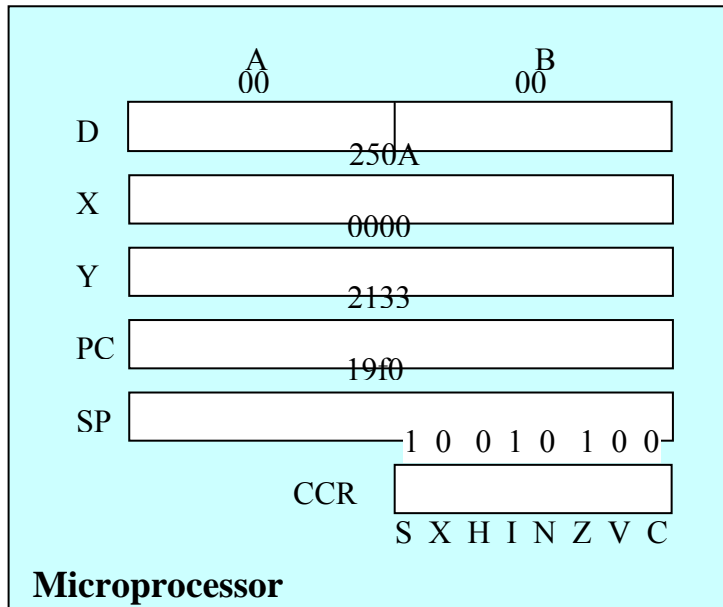
```

```

105: ;-----
106: ; Subroutine: getChBuf
107: ; Return value
108: ;   chr - accumulator A - the character read from the buffer
109: ; Global Variables
110: ;   empty - set to TRUE when buffer is empty
111: ;   bufStart - pointer to start of characters in buffer
112:   2152          getChBuf
113:   2152 34          pshx      ; preserve
114:   2153 FE 2500      ldx bufStart
115:   2156 BE 2502      cpx bufEnd
116:   2159 26 08        bne unbuf
117:   215B F7 2504      tst empty      ; is BUFLen TRUE
118:   215E 27 03        beq unbuf
119:   2160 87          clra      ; chr <- 0
120:   2161 20 12        bra gCBdone
121:   2163 A6 00      unbuf ldaa 0,x      ; get char, inc bufStart
122:   2165 16 2177      jsr incPointer ; increments the pointer in X
123:   2168 7E 2500      stx bufStart
124:   216B BE 2502      cpx bufEnd
125:   216E 26 05        bne gCBdone
126:   2170 180B 01 2504 movb #TRUE,empty ; empty <- TRUE
127:   2175 30          gCBdone pulx      ; restore
128:   2176 3D          rts
129:
130: ;-----
131: ; Subroutine: incPointer(ptr)
132: ; Parameter: ptr - pointer in X register
133:
134:   2177          incPointer:
135:   2177 08          inx      ; increment ptr
136:   2178 8E 2569      cpx #(buffer+BUFLen) ; ptr < buffer+BUFLen ?
137:   217B 2D 03        blt iPend ; Y - branch to end
138:   217D CE 2505      ldx #buffer ; ptr <- buffer
139:   2180 3D          iPend rts
140:
141: ;-----
142: ; ISR: sc0_isr
143: ; Local Variables
144: ;   tchr - accumulator A - character to transmit
145: ; Description:
146: ;   Call getChBuf to obtain next character to transmit.
147: ;   If a null is returned, disable interrupts as the buffer is
148: ;   empty
149:   2181 16 2152      sc0_isr jsr getChBuf ; get char to transmit
150:   2184 97          tsta
151:   2185 27 04        beq sc0IntOff ; Is buffer empty (char is null)
152:   2187 5A CF        staa SC0DRL ; transmit the character
153:   2189 20 03        bra sc0Done
154:   218B          sc0IntOff:
155:   218B 4D CB 40      bclr SC0CR2,%10000000 ; Disable interrupt
156:   218E 0B          sc0Done: rti
157:
158: ***** Data for Driver Module *****
159:   =00002500          ORG DATA
160:   2500 +0002          bufStart ds 2 ; pointer to start of data in buffer
161:   2502 +0002          bufEnd ds 2 ; pointer to end of data in buffer
162:   2504 +0001          empty ds 1 ; number of characters in buffer
163:   2505 +0064          buffer ds BUFLen; the buffer

```

## Annex B – CPU and Memory for Question 1



**IMPORTANT:** The initial conditions correspond to the state of the CPU and memory at the moment the CPU perceives the interrupt from the SC0 interface.

## ANNEX C

### HCS12 INSTRUCTION LIST (reduced)

#### Loads, Stores, and Transfers

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Clear Memory Byte	CLR			X	X	X		m(ea) <= 0
Clear Accumulator A (B)	CLRA (B)						X	A <= 0
Load Accumulator A (B)	LDAA (B)	X	X	X	X	X		A <= [m(ea)]
Load Double Accumulator D	LDD	X	X	X	X	X		D <= [m(ea, ea+1)]
Load Effective Address into SP (X or Y)	LEAS (A,B)							SP <= ea
Store Accumulator A (B)	STAA (B)	X	X	X	X	X		m(ea) <= (A)
Store Double Accumulator D	STD	X	X	X	X	X		m(ea, ea+1) <= D
Transfer A to B	TAB						X	B <= (A)
Transfer A to CCR	TAP						X	CCR <= (A)
Transfer B to A	TBA						X	A <= B
Transfer CCR to A	TPA						X	A <= (CCR)
Exchange D with X (Y)	XGDX						X	D <=> (X)
Pull A (B) from Stack	PULA(B)						X	A <= [m(SP)], SP <= (SP)+1
Push A (B) onto Stack	PSHA(B)						X	SP <= (SP)-1, m(SP) <= A

**Arithmetic Operations**

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Add Accumulators	ABA						X	$A \leftarrow (A) + (B)$
Add with Carry to A (B)	ADCA (B)	X	X	X	X	X		$A \leftarrow (A) + [m(ea)] + (C)$
Add Memory to A (B)	ADDA (B)	X	X	X	X	X		$A \leftarrow (A) + [m(ea)]$
Add Memory to D (16 Bit)	ADDD	X	X	X	X	X		$D \leftarrow (D) + [m(ea, ea+1)]$
Decrement Memory Byte	DEC			X	X	X		$m(ea) \leftarrow [m(ea)] - 1$
Decrement Accumulator A (B)	DECA (B)						X	$A \leftarrow (A) - 1$
Increment Memory Byte	INC			X	X	X		$m(ea) \leftarrow [m(ea)] + 1$
Increment Accumulator A (B)	INCA (B)						X	$A \leftarrow (A) + 1$
Subtract with Carry from A (B)	SBCA (B)	X	X	X	X	X		$A \leftarrow (A) - [m(ea)] - C$
Subtract Memory from A (B)	SUBA (B)	X	X	X	X	X		$A \leftarrow (A) - [m(ea)]$
Subtract Memory from D (16 Bit)	SUBD	X	X	X	X	X		$D \leftarrow (D) - [m(ea, ea+1)]$
Multiply (byte, unsigned)	MUL						X	$D \leftarrow (A) \times (B)$
Multiply word, unsigned (signed)	EMUL(S)						X	$Y:D \leftarrow (D) \times (Y)$
Unsigned (signed) 32 by 16 divide	EDIV(S)						X	$X \leftarrow (Y:D) / (X), Y \leftarrow \text{quotient}, D \leftarrow \text{remainder}$
Fractional Divide ( $D < X$ )	FDIV						X	$X \leftarrow (D) / (X), D \leftarrow \text{remainder}$
Integer Divide (unsigned)	IDIV						X	$X \leftarrow (D) / (X), D \leftarrow \text{remainder}$

**Logical Operations**

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
AND A (B) with Memory	ANDA (B)	X	X	X	X	X		$A \leftarrow A \bullet [m(ea)]$
Bit(s) Test A (B) with Memory	BITA (B)	X	X	X	X	X		$A \bullet [m(ea)]$
One's Complement Memory Byte	COM			X	X	X		$m(ea) \leftarrow \sim [m(ea)]$
One's Complement A (B)	COMA (B)						X	$A \leftarrow \sim A$
OR A (B) with Memory (Exclusive)	EORA (B)	X	X	X	X	X		$A \leftarrow A \oplus [m(ea)]$
OR A (B) with Memory (Inclusive)	ORAA (B)	X	X	X	X	X		$A \leftarrow A + [m(ea)]$

**Shift and Rotate**

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Arithmetic/Logical Shift Left Memory	ASL			X	X	X		
Arithmetic/Logical Left A (B)	ASLA(B)						X	
Arithmetic/Logical Shift Left Double	ASLD						X	
Arithmetic Shift Right Memory	ASR			X	X	X		
Arithmetic Shift Right A (B)	ASRA(B)						X	
Logical Shift Right A (B)	LSRA(B)						X	
Logical Shift Right Memory	LSR			X	X	X		
Logical Shift Right D	LSRD						X	
Rotate Left Memory	ROL			X	X	X		
Rotate Left A (B)	ROLA(B)						X	
Rotate Right A (B)	RORA(B)						X	
Rotate Right Memory	ROR			X	X	X		

**Compare & Test**

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Compare A to B	CBA						X	(A)-(B)
Compare A (B) to Memory	CMPA (B)	X	X	X	X	X		(A) - [m(ea)]
Compare D to Memory (16 Bit)	CPD	X	X	X	X	X		(D) - [m(ea,ea+1)]
Compare SP to Memory (16 Bit)	CPS	X	X	X	X	X		(SP) - [m(ea,ea+1)]
Compare X (Y) to Memory (16 Bit)	CPX	X	X	X	X	X		(X) - [m(ea,ea+1)]
Test memory for 0 or minus	TST			X	X	X		m(ea) - 0
Test A (B) for 0 or minus	TSTA (B)						X	(A)-0

**Short Branches**

Function	Mnemonic	REL	DIR	EXT	IDX	[IDX]	INH	PC <= ea if
Branch ALWAYS	BRA	X						
Branch if Carry Clear	BCC	X						C = 0 ?
Branch if Carry Set	BCS	X						C = 1 ?
Branch if Equal Zero	BEQ	X						Z = 1 ?
Branch if Not Equal	BNE	X						Z = 0 ?
Branch if Higher	BHI	X						Unsigned >
Branch if Lower or Same	BLS	X						Unsigned ≤
Branch if Minus	BMI	X						N = 1 ?
Branch if Plus	BPL	X						N = 0 ?
Branch if Bit(s) Clear in Memory Byte	BRCLR		X	X				[m(ea)]•mask=0
Branch if Bit(s) Set in Memory Byte	BRSET		X	X				[/m(ea)]•mask=0
Branch if Overflow Clear	BVC	X						V = 0 ?
Branch if Overflow Set	BVS	X						V = 1 ?
Branch if Greater Than or Equal	BGE	X						Signed ≥
Branch if Greater Than	BGT	X						Signed >
Branch if Less Than or Equal	BLE	X						Signed ≤
Branch if Less Than	BLT	X						Signed <
Branch if Higher or Same (same as BCC)	BHS	X						Unsigned ≥
Branch if Lower (same as BCS)	BLO	X						Unsigned <
Branch Never	BRN	X						3-cycle NOP

Long branch mnemonic = L + Short branch mnemonic, e.g.: BRA → LBRA

**Loop Primitive Instructions** (counter ctr = A, B, or D)

Function	Mnemonic	REL	DIR	EXT	IDX	[IDX]	INH	Operation
Decrement counter & branch if =0	DBEQ	X						ctr <= (ctr)-1, if (ctr)=0 => PC <= ea
Decrement counter & branch if ≠0	DBNE	X						ctr <= (ctr)-1, if (ctr) ≠0 => PC <= ea
Increment counter & branch if =0	IBEQ	X						ctr <= (ctr)+1, if (ctr)=0 => PC <= ea
Increment counter & branch if ≠0	IBNE	X						ctr <= (ctr)+1, if (ctr) ≠0 => PC <= ea
Test counter & branch if =0	DBEQ	X						if (ctr)=0 => PC <= ea

**Subroutine Calls and Returns**

Function	Mnemonic	REL	DIR	EXT	IDX	[IDX]	INH	Operation
Branch to Subroutine	BSR	X						SP <= (SP)-2, m(SP) <= (PC), PC <= ea
Jump to Subroutine	JSR		X	X	X	X		SP <= (SP)-2, m(SP) <= (PC), PC <= ea
CALL a Subroutine (expanded memory)	CALL		X	X	X	X		SP <= (SP)-2, m(SP) <= (PC), PC <= ea SP <= (SP)-1, m(SP) <= (PPG), PC <= pg
Return from Subroutine	RTS						X	PC <= [m(SP)], SP <= (SP)+2
Return from call	RTC						X	PPG <= [m(SP)], SP <= (SP)+1, PC <= [m(SP)], SP <= (SP)+2

Function	Mnemonic	DIR	EXT	IDX	[IDX]	INH	Operation
Jump	JMP	X	X	X	X		PC <= ea

The **jump** instruction allows control to be passed to any address in the 64-Kbyte memory map.

### Stack and Index Register Instructions

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Decrement Index Register X (Y)	DEX (Y)						X	$X \leftarrow (X) - 1$
Increment Index Register X (Y)	INX (Y)						X	$X \leftarrow (X) + 1$
Load Index Register X (Y)	LDX(Y)	X	X	X	X	X		$X \leftarrow [m(ea, ea+1)]$
Pull X (Y) from Stack	PULX						X	$X \leftarrow [m(SP, SP+1)]$ $SP \leftarrow (SP) + 2$
Push X (Y) onto Stack	PSHX (Y)						X	$m(SP, SP+1) \leftarrow (X)$ $SP \leftarrow (SP) - 2$
Store Index Register X (Y)	STX (X)	X	X	X	X	X		$m(ea, ea+1) \leftarrow X$
Add Accumulator B to X (Y)	ABX (Y)						X	$X \leftarrow (X) + (B)$
Decrement Stack Pointer	DES						X	$SP \leftarrow (SP) - 1$
Increment Stack Pointer	INS						X	$SP \leftarrow (SP) + 1$
Load Stack Pointer	LDS	X	X	X	X	X		$SP \leftarrow [m(ea, ea+1)]$
Store Stack Pointer	STS	X	X	X	X	X		$m(ea, ea+1) \leftarrow (SP)$
Transfer SP to X (Y)	TSX (Y)						X	$X \leftarrow (SP)$
Transfer X (Y) to SP	TXS (Y)						X	$SP \leftarrow (X)$
Exchange D with X (Y)	XGDX (Y)						X	$(D) \leftrightarrow (X)$

Function	Mnemonic	INH	Operation
Return from Interrupt	RTI	X	$(M_{(SP)} \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow SP$
Software Interrupt	SWI	X	
Wait for Interrupt	WAI	X	

### Interrupt Handling

The software interrupt (SWI) instruction is similar to a JSR instruction, except the contents of all working CPU registers are saved on the stack rather than just the return address. SWI is unusual in that it is requested by the software program as opposed to other interrupts that are requested asynchronously to the executing program



## **Annex C - Register and Bit Definitions for 9S12DG256 Peripheral Modules**

DATA REGISTERS								CONTROL REGISTERS								STATUS REGISTERS							
PARALLEL PORTS								DDRA \$0002    DDRB \$0003    DDRE \$0009    DDRK \$0033															
PORTA \$0000    PORTB \$0001    PORTE \$0008    PORTK \$0032								DDRA7 DDRA6    DDRA5 DDRA4    DDRA3 DDRA2    DDRA1 DDRA0															
PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0								DDRB7 DDRB6    DDRB5 DDRB4    DDRB3 DDRB2    DDRB1 DDRB0															
PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0								DDRE7 DDRE6    DDRE5 DDRE4    DDRE3 DDRE2    DDRE1 DDRE0															
PE7 PE6 PE5 PE4 PE3 PE2 PE1 PE0								DDRK7 DDRK6    DDRK5 DDRK4    DDRK3 DDRK2    DDRK1 DDRK0															
PK7 PK6 PK5 PK4 PK3 PK2 PK1 PK0								PUCR \$000C															
								PUPKE 0    0    PUPPE 0    0    PUPBE PUPAE															
TIMER								TIOS \$0040    Timer								TFLG1 \$004E    Timer Interrupt Flag Register 1							
TCNT \$0044    \$0045    Timer Count Register								Input-Capture (IOS=0)/Output-Compare (IOS=1) Select Register								C7F C6F C5F C4F C3F C2F C1F C0F							
Bit 15 Bit 14 Bit 13 Bit 12 Bit 11 Bit 10 Bit 9 Bit 8								IOS7 IOS6 IOS5 IOS4 IOS3 IOS2 IOS1 IOS0								TFLG2 \$004F    Timer Interrupt Flag 2							
Bit 7 Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 Bit 0								CFORC \$0041    Timer Compare Force Register								TOF 0 0 0 0 0 0 0 0							
								FOC7 FOC6 FOC5 FOC4 FOC3 FOC2 FOC1 FOC0															
								OC7M \$0042    Timer Output Compare 7 Mask															
								OC7M7 OC7M6 OC7M5 OC7M4 OC7M3 OC7M2 OC7M1 OC7M0															
								OC7D \$0043    Timer Ouput Compare 7 Data Register															
								OC7D7 OC7D6 OC7D5 OC7D4 OC7D3 OC7D2 OC7D1 OC7D0															
								TCTL1/TCTL2 \$0048/49    Timer Control Register 1/2															
								OM7 OL7 OM6 OL6 OM5 OL5 OM4 OL4															
								OM3 OL3 OM2 OL2 OM1 OL1 OM0 OL0															
								TCTL3/TCTL4 \$004A/4B    Timer Control Register 4															
								EDG7B EDG7A EDG6B EDG6A EDG5B EDG5A EDG4B EDG4A															
								EDG3B EDG3A EDG2B EDG2A EDG1B EDG1A EDG0B EDG0A															
								TSCR1 \$0046    Timer System Control Register 1															
								TEN TSWAI TSFRZ TFFCA 0 0 0 0															
								TSCR2 \$004D    Timer System Control Register 1															
								TOI 0 0 0 0    TCRE PR2 PR1 PR0															
								TIE \$004C    Timer Interrupt Enable Register															
								C7I C6I C5I C4I C3I C2I C1I C0I															
SERIAL INTERFACE-SCI								SC0BDH \$00C8    \$00C9    SCI Baud Rate Control Register								Baud Rate SER    PT=0 => even							
SC0DRH \$00CE    SCI Data Register High								0 0 0 SBR12 SBR11 SBR10 SBR9 SBR8								2400 651 M=0 => 1 start bit, 8 data bits, 1 stop bit							
R8 T8 0 0 0 0 0 0								SBR7 SBR6 SBR5 SBR4 SBR3 SBR2 SBR1 SBR0								9600 163 RDRF = 0 => SC0DR is empty							
SC0DRL \$00CF    SCI Data Register Low								SC0CR1 \$00CA    SCI Control Register 1								19,200 81 \$FFD6 = SC10 interrupt vector address							
R7T7 R6T6 R5T5 R4T4 R3T3 R2T2 R1T1 R0T0								LOOPS SCISWAI RSRC M WAKE ILT PE PT								38,400 41							
								SC0CR2 \$00CB    SCI Control Register 2								SC0SR1 \$00CC    SCI Status Register 1							
								TIE TCIE RIE ILIE TE RE RWU SBK								TDRE TC RDRF IDLE OR NF FE PF							
																SC0SR2 \$00CD    SCI Status Register 2							
																0 0 0 0 0 0 BRK13 TXDIR RAF							
SERIAL INTERFACE-SPI								SP0BR \$00DA    SPI Baud Rate Control Register								E-Clock Divisor = ((SPPR+1)*2)^SPPR for SPPR > 0							
SP0DR \$00Dd    SPI Data Register								0 SPPR2 SPPR1 SPPR0 0 SPR2 SPR1 SPR0															
Bit 7 Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 Bit 0								SP0CR1 \$00D8    SPI Control Register 1															
								SPIE SPE SPTIE MSTR CPOL CPHA SSOE LSBF															
								SP0CR2 \$00D9    SPI Control Register 2								SP0SR \$00DB    SPI Status Register							
								0 0 0 MODFEN BIDRIOE 0 SPISWAI SPC0								SPIF 0 SPTIE MODF 0 0 0 0							
								ATDCTL2 \$82    ATD Control Register 2								ATDSTAT0 \$86    ATD Status Register 0							
								ADPU AFFC AWAI ETRIGLE ETRIGP ETRIG ASCIE ASCIF								SCF 0 ETORF FIFOR 0 CC2 CC1 CC0							
ANALOG-TO-DIGITAL SUBSYSTEM								ATDCTL3 \$83    ATD Control Register 3								ATDSTAT1 \$86    ATD Status Register 1							
ADRxH \$90+2x    ATD Result Registers High								0 S8C S4C S2C S1C FIFO FRZ1 FRZ2								CCF7 CCF6 CCF5 CCF4 CCF3 CCF2 CCF1 CCF0							
Bit 15 Bit 14 Bit 13 Bit 12 Bit 11 Bit 10 Bit 9 Bit 8								ATDCTL4 \$84    ATD Control Register 4															
ADRxL: \$91+2x    ATD Result Registers Low								SRES8 SMP1 SMP0 PRS4 PRS3 PRS2 PRS1 PRS0															
Bit 7 Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 Bit 0								ATDCTL5 \$85    ATD Control Register 5															
x=0...7								DJM DSGN SCAN MULT 0 CC CB CA															
								INTCR \$001E    Interrupt Control Register								HPRIO \$001F    Highest Priority I Interrupt Register							
								IRQE IRQEN DLY 0 0 0 0 0 0								PSEL7 PSEL6 PSEL5 PSEL4 PSEL3 PSEL2 PSEL1 0							

## Timer Module

- TEN - Activate (=0) and deactivate (=1) timer
- AFFC - allows automatic clear of flags (=1)
- TOI enable (=1) overflow interrupt (from \$FFFF to \$0000) of the TCNT
- PR2, PR1, PR0 prescale bits ( $= 2^{PR}$ ) applied to system clock for incrementing timer
- IOS0 to IOS7 – Configure channel as output compare (=1) or input-capture (=0)
- FOC0-FOC1 – Force output-compare event on channel
- OC7M0-OC7M7 – Allow channel 7 to affect channel pin (=1)
- OC7D0-OC7D7 – Value of level for channel 7 to output to channel pin
- C0I-C7I – Enable a channel interrupt
- C0F-C7F – Channel flag set when event occurs
- TOF – Counter Overflow flag

## ATD Module

- ADPU enable (=1) converter
- AFFC enable (=1) fast clear
- ASCIE enable (=1) interruption at the completion of a conversion sequence
- ASCFI flags (=1) a sequence conversion
- S8C,S4C,S2C,S1C defines number of conversions per sequence (when set to 0, or S8C=1,8 conversions in a sequence)
- FIFO – Use FIFO discipline to fill registers when set to 1 (otherwise fill in starting at register ADR0).
- SRES8 selects resolution (1 = 8 bits, 0 = 10 bits)
- SMP1, SMP2 – sample time ( $= 2^{SMP+1}$ )
- PRS4, PRS3, PRS2, PRS1, PRS0 – prescaler of system bus clock (of frequency BUS = – typically 24 MHz) to generate ATD clock (of frequency CC):  

$$CC = (BUS / (2 * (PRS + 1)))$$
, where PRS is value represented by PRS4 to PRS0.
- DJM – Right justify (=1) or left justify (=0) results
- DSGN – Used signed (=1) or unsigned numbers (=0) (significant only if DJM=0)
- SCAN – single conversion (=0) or scan mode (=1)
- MULT – single pin (=0) or multiple pins (=1)
- CC, CB, CA – Selects pin for single conversion or first pin in the case of multiple conversion.

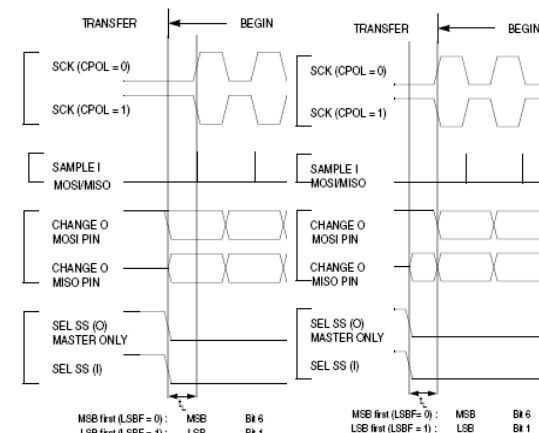
ATD Cycles per conversion  
 2 (initial sample time) + 2 to 16 (final sample time) + 8 or 10 (conversion time for 8 bits or 10 bits).

## Serial Interface Module – SCI subsystem

- M: Two modes 8 data bits or 9 data bits (with 1 start bit and 1 stop bit)
- PE, PT: Enable parity (PE) and type of parity (PT)
- TIE, TCIE, RIE: Bits for enabling interruptions for reception and transmission
- TE, RE: Enable transmission (TE) and reception (RE)
- TDRE: 1 indicates that more data can be written into the data register
- TC: 1 indicates that transmission of data is complete
- RDRF: 1 indicates that reception data register is full
- OR, NF, FE, PF: error bits, equal 1 when an error is detected during reception (overrun, noise, framing, parity)
- RAF: 1 indicates that reception of a character is underway.

## Serial Interface Module – SPI subsystem

- LSBF – Least significant bit first (0 – MSB)
- SS0E – Slave Select
- CPHA et CPOL – see diagrams to the right
- MSTR – Master mode (1) or slave mode (0)
- SPTIE – Enable transmit interrupt (SPTEF)
- SPE – Enable SPI interface
- SPIE – Enable SPI interface interrupts (SPIF/MODF)
- SPC0 (bit 0) – Defines pin configuration with MSTR
- BIDRIOE – Set to enable output buffer
- MODFEN – Enable Modf feature in Master
- SPIF – Set at the completion of data transfer (8 clock cycles), clear by read of SPISR and access to SPIDR.
- SPTEF – Set to indicate that transmission data register is empty
- MODF – Set when SS configured in Master for error detection (input)



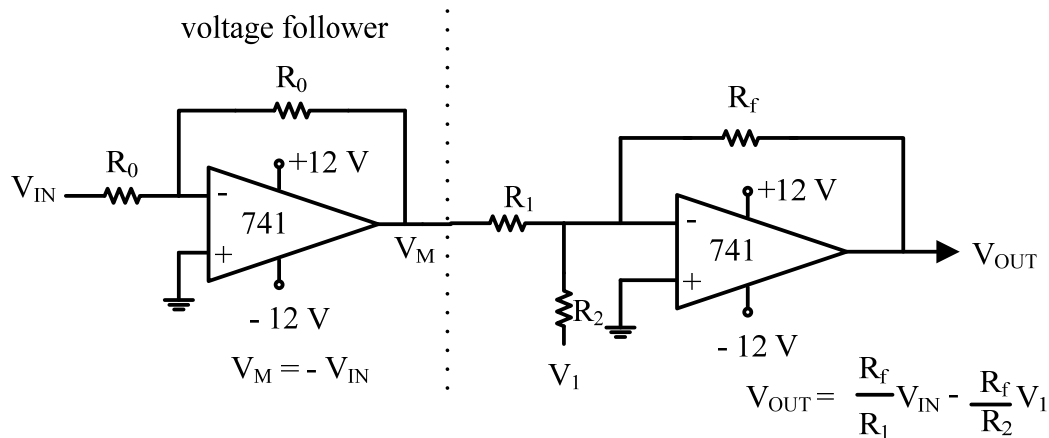
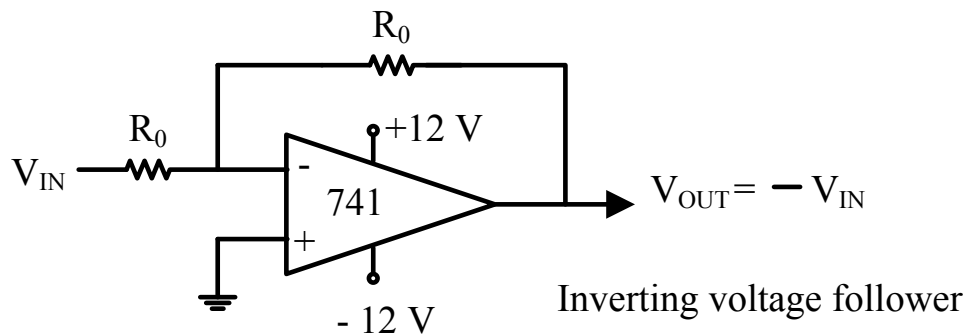
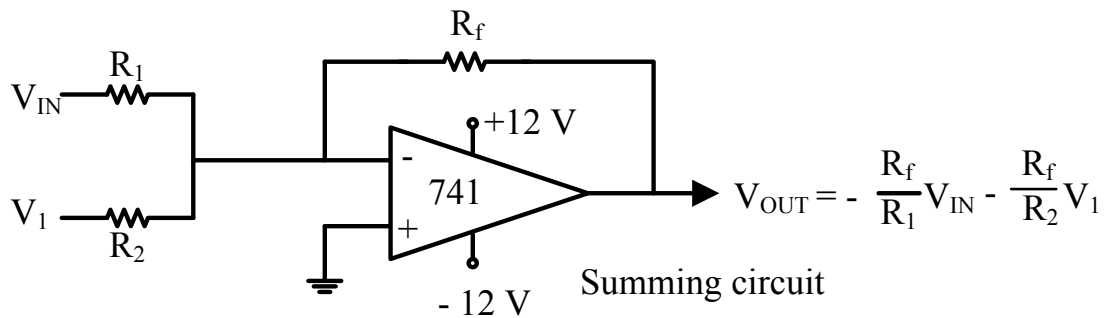
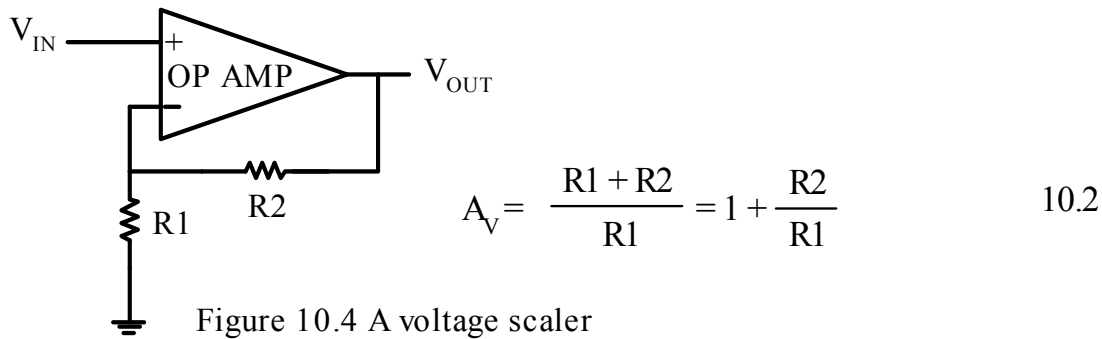
CPHA=0

CPHA=1

Table 3-3 Bidirectional Pin Configurations

Table 1: Bidirectional Pin Configurations				
Pin Mode	SPC0	BIDRIOE	MISO	MOSI
Normal	0	X	Master In	Master Out
Bidirectional	1	0	MISO not used by SPI	Master In
		1		Master I/O
Slave Mode of Operation				
Normal	0	X	Slave Out	SlaveIn
Bidirectional	1	0	Slave In	MOSI not used by SPI
		1	Slave I/O	

## Annex D Conditioning Circuits



Standard resistor values: 10,12,15,18,22,27,33,39,47,56,68,82 (multiply by multiples of 10 to get values between 10 ohms and 10 Mohms)