@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ ECE 372 – PROGRAMMING PROJECT III

@--------------------------------------------------------------------------------

@ Program implement Clock with real time. At the beginning program, it show my name

@ After pressing Button, The real time clock appears. Furthermore, User can use

@ switch to reset value of real time clock and control Brightness of backlight

@ Program combine TIMER + PWM + I2C.

@-------------------------------------------------------------------------------

@ Hai Dang Hoang

@ Reference: Douglas V. Hall and Leela Yadlapalli

@ March 21th, 2015

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

@-------------------------------INITIALIZE STACK--------------------------------@

LDR R13,=STACK1 @ Point to base of STACK for SVC mode

ADD R13, R13,#0X1000 @ Point to top of STACK

CPS #0x12 @ Switch to IRQ mode

LDR R13,=STACK2 @ Point to IRQ stack

CPS #0x13 @ Back to SVC mode

LDR R0,=0x4804C000 @ Base address for GPIO1 registers

ADD R4, R0,#0x190 @ Address of GPIO1\_CLEARDATAOUT register

MOV R7, #0x01E00000 @ Load value to turn off Rotate\_LED on GPIO1\_21,22,23,24

STR R7,[R4] @ Write to GPIO1\_CLEARDATAOUT register

@-------------------------------SET UP FOR GPIO---------------------------------@

ADD R1,R0,#0x0134 @ Get GPIO1\_OE register address

LDR R6,[R1] @ READ current GPIO1 Output Enable register

LDR R7,=0xFE1FFFFF @ Word to enable GPIO1\_21,22,23,24 as output

AND R6,R7,R6 @ Clear bit 12 (MODIFY)

STR R6, [R1] @ WRITE to GPIO1 Output Enable register

@----------------------------INITIALIZE FALLING EDGE DECTECT------------------@

ADD R1, R0, #0x14C @ Falling edge detect

LDR R2, =0x80000000 @ Detect by GPIO1\_31

LDR R3, [R1] @ Read current value in GPIO1\_FALLING\_EDGE\_DETECT

ORR R3, R3, R2 @ Modify the current value

STR R3, [R1] @ Write modified value back to GPIO1\_FALLINGEDGEDECTECT

ADD R1, R0, #0x34 @ Get address of GPIO1\_IRQSTATUS\_SET0

STR R2, [R1] @ Enable GPIO1\_31 request on GPIO1\_IRQSTATUS\_SET0

@-----------------------------Initialize INTC-----------------------------------@

@ Reset

LDR R1, =0x48200010 @ INTC\_SYSCONFIG

MOV R2,#2 @ Value for reset

STR R2,[R1] @ Software Reset

@ Enable timer interrupt

LDR R1,=0x482000C8 @ Address of INTC\_MIR\_CLEAR2 register

MOV R2, #0x10 @ Value to unmask INTC INT 68, DMTIMER2, POINTR\_PEND

STR R2, [R1] @ Write to INTC\_MIR\_CLEAR2 register

@ Enable GPIO interrupt

LDR R1, =0x482000E8 @ Address INTC\_MIR\_CLEAR3

MOV R2, #0x00000004 @ Value to unmask INTC INT 98, GPIO1, POINTRPEND1

STR R2, [R1] @ Write to INTC\_MIR\_CLEAR3 register

@---------------------------SET UP FOR TIMER-------------------------------------@

@ Enable Timer 2 Clock

LDR R1,=0x44E00080 @ Address of CM\_PER\_TIMER2\_CLKCTRL

MOV R2, #2 @ Value to enable the timer 2 clock

STR R2, [R1] @ Enable Timer 2 clock

@ Initialize Timer 2

LDR R1,=0x48040000 @ Base address of Timer 2

@ Reset

MOV R2,#1 @ Value for reset

STR R2,[R1,#0x10] @ Store Timer Config ( TIOCP\_CFG)

@ Prime the timer

LDR R2, =0xFFFF8300 @ 1 second count from 0xFFFF8300 to 0xFFFFFFFF

STR R2, [R1,#0x3C] @ Store it into Timer TCRR

LDR R2, =0xFFFF8300 @ 1 second count from 0xFFFF8300 to 0xFFFFFFFF

STR R2, [R1,#0x40] @ Store it into Timer TLDR

@ Start Auto-reload timer

MOV R2,#3 @ Auto-reload, start

STR R2,[R1,#0x38] @ Store it in Time TCLR

@ --------------------------------Initialize PWM----------------------------------@

@ Enable PWM

LDR R0,=0x44e000CC @ Address CM\_PER-EPWMSS1\_CLKCTRL

MOV R1,#0x2 @ Enable Clock for PWM

STR R1,[R0]

@ Control Module For PWM

LDR R0,=0x44e10664 @ Address of pwmss\_ctrl

MOV R1,#0x2 @ Select Timebase clock enable for PWMSS1

STR R1,[R0]

@ Pin PWM output ( Pin Mux)

LDR R0,=0x44e10848 @ Address of conf\_gpmc\_a1

MOV R1,#0x6 @ Choose Mode 6

STR R1,[R0]

@ Initial PWM

/\* Disable PWM stopping on debug events \*/

LDR R0,=0x48302200 @ Address Time-Base Control Register

LDR R1,=0xC000 @ Select Time-base Clock is divide 1

@ Select Up-count mode

@ Select Free run

STRH R1,[R0]

/\* Set period \*/

LDR R0,=0x4830220A @ Address Time-Base Period Register

LDR R1,=0xC350 @ Select Value 0xC350 for Time-base period

STRH R1,[R0]

/\* Set compare A value of the period to get duty cycle \*/

LDR R0,=0x48302212 @ Address Counter-Compare A Register

LDR R1,=0x186 @ Select 2.5% duty cycle

STRH R1,[R0]

/\* Configure output \*/

LDR R0,=0x48302216 @ Output A (EPWMxA)

MOV R1,#0x1A @ Select force EPWM xA output high

@ When counter equals active CMPA, counter increase

STRH R1,[R0]

LDR R4,=value @ R4 Register hold memory store value of duty cycle

MOV R5,#0x0 @ R5 Register is pointer control memory

@---------------------------------ENABLE CLOCK FOR I2C1-----------------------------@

LDR R0,=0x44E00048 @ Enable Clock for I2C1 (CM\_PER\_I2C1\_CLKCTRL)

MOV R1,#0x2 @ Store 2 to enable

STR R1,[R0]

@--------------------------------CONTROL MODULE I2C1--------------------------------@

LDR R8,=0x44e10958 @ use Pin 17 on P9 for SCL

MOV R9,#0x72 @ Slow slew rate/ Pull Up/ Receiver enable/ Mode 2

STR R9,[R8] @ Store

LDR R8,=0x44e1095C @ use Pin 18 on P9 for SDA

MOV R9,#0x72 @ Slow slew rate/ Pull Up/ Receiver enable/ Mode 2

STR R9,[R8] @ Store

@--------------------------------INITIAL I2C1---------------------------------------@

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x0 @ Disable I2C\_EN by Storing 0

STR R1,[R0]

LDR R0,=0x4802A010 @ LOAD System Configuration Register

MOV R1,#0x0 @ Disable auto Idle

STR R1,[R0]

@ Configure Frequency = 100 KHz

@ Frequency = ICLK / ((SCLL + 7) + (SCLH + 5))

@ ICLK = 48 MHz / I2C\_PSC.PSC

@ Choose PSC = 4 --> ICLK = 48/4 = 12MHz

@ I2C\_SCLL.SCLL = I2C\_SCLH.SCLH ( Technical Reference Manual)

@ Solving SCLL = SCLH = 0x36

@ Setting PSC 3 for divide/4

LDR R0,=0x4802A0B0 @ Load I2C Clock Prescaler Register

MOV R1,#0x3 @ Divide 4

STR R1,[R0]

@ Setting SCLL and SCLH : 0x36

LDR R0,=0x4802A0B4 @ Load I2C\_SCLL Register

MOV R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A0B8 @ Load I2C\_SCLH register

MOV R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A094 @ Load Buffer Configuration Register

MOV R1,#0x0 @ RXTRSH/TXTRSH = 1

STR R1,[R0]

LDR R0,=0x4802A0A4 @ I2C Configuration Register

LDR R1,=0x8000 @ Enable I2C\_EN

STR R1,[R0]

@---------------------------------LCD\_INIT------------------------------------------@

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 1 to clear all

STR R1,[R0]

@ Sending Slave address 0x3C and along with 10 data bytes

MOV R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

MOV R1,#10 @ 10 Data bytes ( Counter)

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C Enable/Master/Transmit/STT/STP

STR R1,[R0]

@ Sending the first packet data: 0x0

MOV R1,#0x0 @ 0x0: Control Byte

BL Transmit\_Data

@ Sending the second byte data: 0x38

MOV R1,#0x38 @ 0x38: Func set

BL Transmit\_Data

@ Sending the third byte data: 0x39

MOV R1,#0x39 @ 0x39 : Func set

BL Transmit\_Data

@ Delay 10ms

BL Delay

@ Sending the fourth byte data: 0x14

MOV R1,#0x14 @ 0x14: Bias set

BL Transmit\_Data

@ Sending the fifth byte data: 0x78

MOV R1,#0x78 @ 0x78: Contrast set

BL Transmit\_Data

@ Sending the sixth byte data: 0x5E

MOV R1,#0x5E @ 0x5E: Power/ICON control/ Contrast set

BL Transmit\_Data

@ Sending the seventh byte data: 0x6D

MOV R1,#0x6D @ 0x6D: Follower control

BL Transmit\_Data

@ Sending the eighth byte data: 0x0C

MOV R1,#0x0C @ 0x0C: Display on

BL Transmit\_Data

@ Sending the ninth byte data: 0x01

MOV R1,#0x01 @ 0x01: Clear Display

BL Transmit\_Data

@ Sending the tenth byte data: 0x06 ( last data byte)

MOV R1,#0x06 @ 0x06: Entry mode set

BL Transmit\_Data

@ Wait access

BL Wait\_access

BL Turn\_off\_ardy

@ Delay 10ms

BL Delay

@-------------------------------DISPLAY MY NAME------------------------------------@

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 1 to clear all

STR R1,[R0]

@ Sending Slave address 0x3C and Data bytes

MOV R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

MOV R1,#12 @ 10 Data bytes ( Counter)

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C Enable/Master/Transmit/STT/STP

STR R1,[R0]

@ Sending the first data: 0x80

MOV R1,#0x80 @ 0x80: Control byte

BL Transmit\_Data

@ Sending the second byte data: 0x86

MOV R1,#0x56 @ 0x86: Position

BL Transmit\_Data

@ Sending the third byte data: 0x40

MOV R1,#0x40 @ 0x40: Datasend

BL Transmit\_Data

@ Sending the fourth byte data: 0x48

MOV R1,#0x48 @ ( Character: H)

BL Transmit\_Data

@ Sending the fifth byte data: 0x61

MOV R1,#0x61 @ ( Character: a)

BL Transmit\_Data

@ Sending the sixth byte data: 0x69

MOV R1,#0x69 @ ( Character: i)

BL Transmit\_Data

@ Sending the space byte data: 0x20

MOV R1,#0x20 @ ( Character: space)

BL Transmit\_Data

@ Sending the seventh byte data: 0x44

MOV R1,#0x44 @ ( Character: D)

BL Transmit\_Data

@ Sending the eighth byte data: 0x61

MOV R1,#0x61 @ ( Character: a)

BL Transmit\_Data

@ Sending the ninth byte data: 0x6E

MOV R1,#0x6E @ ( Character: n)

BL Transmit\_Data

@ Sending the tenth byte data: 0x67

MOV R1,#0x67 @ ( Character: g)

BL Transmit\_Data

@ Sending the eleventh byte data: 0x3A

MOV R1,#0x3A @ ( Character: ':')

BL Transmit\_Data

@ Wait Access

BL Wait\_access

BL Turn\_off\_ardy

@ Make sure processor IRQ enabled in CPSR

MRS R3, CPSR @ Copy CPSR to R3

BIC R3,#0x80 @ Clear bit 7

MSR CPSR\_c, R3 @ Write back to CPSR

**LOOP:**

NOP

B LOOP

@--------------------------INTERRUPT------------------------------------------@

**INT\_DIRECTOR:**

STMFD SP!,{R0-R3,LR} @ Push registers on stack

LDR R0,=0x482000F8 @ Address of INTC-PENDING\_IRQ3 register

LDR R1,[R0] @ Read INTC-PENDING\_IRQ3 register

TST R1,#0x00000004 @ TEST BIT 2

BEQ Check\_TIMER\_interrupt @ If interrupt doesn't come from GPIO1

LDR R0,=0x4804C02C @ Load GPIO1\_IRQSTATUS\_0 register address

LDR R1,[R0] @ Read Status register

TST R1,#0x80000000 @ Check if bit 31=1

BNE BUTTON\_SVC @ If bit 31=1, then button pushed

BEQ PASS\_ON @ If bit 31=0, then go to back to wait loop

**Check\_TIMER\_interrupt:**

@ Check that this is the timer interrupt

LDR R0,=0x482000D8 @ Address of INTC-PENDING\_IRQ2 register

LDR R1,[R0] @ Read INTC-PENDING\_IRQ2 register

TST R1,#0x10 @ TEST BIT 4

BEQ PASS\_ON @ Interrupt doesn't come from Timer 2

@ Check that this is the overflow interrupt

LDR R0, =0x48040028 @ Address of Timer 2 IRQ Status

LDR R1,[R0] @ Load value from

CMP R1, #2 @ Check Overflow

BEQ IRQ\_timer @ If overflow -> change state of Rotate\_LED

**PASS\_ON:**

LDMFD SP!,{R0-R3,LR} @ Restore registers

SUBS PC,LR,#4 @ Pass execution on to wait LOOP for now

@---------------------------Timer Overflow-----------------------------

**IRQ\_timer:**

@ Clear timer interrupts

LDR R1, =0x48040028 @ Timer2\_IRQ Status (IRQSTATUS)

MOV R2, #2 @ Capture, overflow , and match

STR R2, [R1] @ Write 1 to clear interrupt

@ Generate new IRQ generation

LDR R0,=0x48200048 @ Address of INTC\_CONTROL register

MOV R1, #01 @ Value to enable new IRQ generation

STR R1,[R0] @ Write to INTC\_CONTROL register

@ Update I2C display on Screen when TIMER overflow

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 1 to clear all

STR R1,[R0]

@----------------------- Call Procedure Update Clock-----------------------

BL UPDATE\_CLOCK

@----------------------------I2C Sending---------------------------------------

@ Sending Slave address 0x3C and Data bytes

MOV R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

MOV R1,#12 @ 8 Data bytes ( Counter)

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C Enable/Master/Transmit/STT/STP

STR R1,[R0]

@ Sending the first data: 0x80

MOV R1,#0x80 @ 0x80: Control byte

BL Transmit\_Data

@ Sending the second byte data: 0x86

MOV R1,#0x89 @ Position cursor

BL Transmit\_Data

MOV R1,#0x40 @ Data send

BL Transmit\_Data

@ Sending the space data: 0x20

MOV R1,#0x20 @ 0x20: Space

BL Transmit\_Data

@ Sending the third byte data:

LDR R0,=tenth\_hour @ Address Tenth\_hour

LDR R1,[R0] @ Get value from Tenth\_hour

BL Transmit\_Data

@ Sending the fourth byte data:

LDR R0,=unit\_hour @ Address Unit\_hour

LDR R1,[R0] @ Get value from unit\_hour

BL Transmit\_Data

MOV R1,#0x3A @ ( Character: ':')

BL Transmit\_Data

@ Sending the fifth byte data:

LDR R0,=tenth\_minute @ Address Tenth\_minute

LDR R1,[R0] @ Get value from Tenth\_minute

BL Transmit\_Data

@ Sending the sixth byte data:

LDR R0,=unit\_minute @ Address Unit Minute

LDR R1,[R0] @ Get value from Unit\_Minute

BL Transmit\_Data

@ Sending the seventh byte data: 0x3A

MOV R1,#0x3A @ ( Character: ':')

BL Transmit\_Data

@ Sending eighth byte data:

LDR R0,=tenth\_second @ Address Tenth\_Second

LDR R1,[R0] @ Get value of Tenth\_second

BL Transmit\_Data

@ Sending ninth byte data:

LDR R0,=unit\_second @ Address Unit\_second

LDR R1,[R0] @ Get value of Unit\_second

BL Transmit\_Data

@ Wait Access

BL Wait\_access

BL Turn\_off\_ardy

**DONE:**

LDMFD SP!, {R0-R3,LR} @ Restore registers

SUBS PC, LR, #4 @ Pass execution on to wait LOOP for now

@ ----------------------------Procedure UPDATE\_CLOCK ------------------------------@

@ UPDATE\_CLOCK procedure is updated after 1 second from timer

**UPDATE\_CLOCK:**

STMFD SP!,{R0-R3,LR} @ Push registers on stack

LDR R0,=unit\_second @ Load Memory Unit\_second

LDR R1,[R0] @ Get value from Unit\_second

CMP R1,#0x39 @ Check Unit\_second equals '9'

ADDNE R1,R1,#0x1 @ If no, Increment Unit\_second by 1

STRNE R1,[R0] @ Store into memory Unit\_second

BNE Finish @ Finish update !

@ Else ( Currently, Unit\_second == '9' -> unit\_second = '0' and...)

MOV R1,#0x30 @ Update Unit\_second = 0

STR R1,[R0] @ Store into memory Unit\_second

LDR R2,=tenth\_second @ Load Memory Tenth\_Second

LDR R3,[R2] @ Get value from Tenth\_Second

CMP R3,#0x35 @ Check Tenth\_second equals '5'

ADDNE R3,R3,#0x1 @ If no, Increment Tenth\_second by 1

STRNE R3,[R2]

BNE Finish @ Finish Update !

@ Else ( Currently, Tenth\_second == '5' -> Tenth\_second = '0' and ...)

MOV R3,#0x30 @ Update Tenth\_Second = 0

STR R3,[R2] @ Store Tenth\_Second into memory

LDR R0,=unit\_minute @ Load address Unit\_minute

LDR R1,[R0] @ Load value from Unit\_minute

CMP R1,#0x39 @ Check Unit\_Minute equals '9'

ADDNE R1,R1,#0x1 @ If no, Increment Unit\_minute by 1

STRNE R1,[R0] @ Store into memory Unit\_minute

BNE Finish @ Finish

@ Else ( Currently, Unit\_minute == '9' -> Unit\_minute = '0' and ...)

MOV R1,#0x30 @ Update Tenth\_Minute = 0

STR R1,[R0] @ Store Tenth\_Minute into memory

LDR R2,=tenth\_minute @ Load address from Tenth\_minute

LDR R3,[R2] @ Load value from Address

ADD R3,R3,#0x1 @ Increment Tenth\_Minute by 1

STR R3,[R2] @ Store value into Tenth\_Minute

**Finish:**

LDMFD SP!, {R0-R3,LR} @ Restore registers

MOV PC, R14

@-----------------------------Procedure BACK\_UP\_CLOCK-------------------------------@

@ Procedure BACK\_UP\_CLOCK will reset all value in memory of

@ Tenth\_hour, Unit Hour, Tenth\_Minute, Unit\_Minute, Tenth\_Second, Unit\_Second

@ It has form: 00:00:00

**BACK\_UP\_CLOCK:**

STMFD SP!,{R0-R3,LR} @ Push registers on stack

MOV R1,#0x30 @ Character '0'

LDR R0,=tenth\_hour @ Back up tenth\_hour

STR R1,[R0]

LDR R0,=unit\_hour @ Back up unit\_hour

STR R1,[R0]

LDR R0,=tenth\_minute @ Back up tenth\_minute

STR R1,[R0]

LDR R0,=unit\_minute @ Back up unit minute

STR R1,[R0]

LDR R0,=tenth\_second @ Back up tenth\_second

STR R1,[R0]

LDR R0,=unit\_second @ Back up unit\_second

STR R1,[R0]

LDMFD SP!, {R0-R3,LR} @ Restore registers

MOV PC, R14

@--------------------------Button Service------------------------------------@

**BUTTON\_SVC:**

LDR R1,=0x80000000 @ Value to turn off GPIO1\_31 Interrupt request

@ This will turn off INTC interrupt request also

STR R1,[R0] @ Write to GPIO1\_IRQSTATUS\_0 register

@ Turn off NEWIRQA bit in INTC\_CONTROL, so processor can respond to new IRQ

LDR R0,=0x48200048 @ Address of INTC\_CONTROL register

MOV R1,#0x1 @ Value to clear bit 0

STR R1,[R0] @ Write to INTC\_CONTROL register

@----------------------------Enable Timer Interrupt---------------------------@

LDR R0, =0x4804002C @ Address IRQENABLE\_SET

MOV R1,#0x2 @ Enable Timer interrupt

STR R1,[R0]

@----------------------------------- Control PWM------------------------------@

CMP R5,#40 @ Check Pointer is out of 40

MOVEQ R5,#0x0 @ If yes 🡪 Reset R5=0

ADDNE R5,#0x4 @ If No 🡪 Increment pointer by 4

/\* Set period again after pressing Button \*/

LDR R0,=0x48302212 @ Address Counter-Compare A Register

LDR R1,[R4,R5] @ Get value duty cycle from Memory

STRH R1,[R0] @ Store

@-----------------------------TIME: 00:00:00----------------------------------@

BL BACK\_UP\_CLOCK @ Back up clock 00:00:00

@-----------------------------------------------------------------------------@

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 1 to clear all

STR R1,[R0]

@ Sending Slave address 0x3C and Data bytes

MOV R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

MOV R1,#12 @ 12 Data bytes ( Counter)

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C Enable/Master/Transmit/STT/STP

STR R1,[R0]

@ Sending the first data: 0x80

MOV R1,#0x80 @ 0x80: Control byte

BL Transmit\_Data

@ Sending the second byte data: 0x86

MOV R1,#0x89 @ Position cursor

BL Transmit\_Data

@ Sending the third byte data: 0x40

MOV R1,#0x40 @ 0x40: Datasend

BL Transmit\_Data

@ Sending the space data: 0x20

MOV R1,#0x20 @ 0x20: Space

BL Transmit\_Data

@ Sending the fourth byte data: 0x30

MOV R1,#0x30 @ ( Character: 0)

BL Transmit\_Data

@ Sending the fifth byte data: 0x30

MOV R1,#0x30 @ ( Character: 0)

BL Transmit\_Data

@ Sending the sixth byte data: 0x3A

MOV R1,#0x3A @ ( Character: ':')

BL Transmit\_Data

@ Sending the space byte data: 0x30

MOV R1,#0x30 @ ( Character: 0)

BL Transmit\_Data

@ Sending the seventh byte data: 0x30

MOV R1,#0x30 @ ( Character: 0)

BL Transmit\_Data

@ Sending the eighth byte data: 0x3A

MOV R1,#0x3A @ ( Character: ':')

BL Transmit\_Data

@ Sending the ninth byte data: 0x30

MOV R1,#0x30 @ ( Character: 0)

BL Transmit\_Data

@ Sending the tenth byte data: 0x30

MOV R1,#0x30 @ ( Character: 0)

BL Transmit\_Data

@ Wait Access

BL Wait\_access

BL Turn\_off\_ardy

**Done:**

LDMFD SP!,{R0-R3,LR} @ Restore registers

SUBS PC,LR,#4 @ Return from IRQ interrupt procedure

@ --------------------PROCEDURE: Transmit 1 byte data from BBB to I2C\_LCD-----------@

**Transmit\_Data:**

STMFD R13!,{R0,R14} @ Store uses registers on Stack

LDR R0,=0x4802A09C @ Load I2C Data Register

STR R1,[R0] @ Store value to transmitting

@ Wait xrdy

BL Wait\_transmit

BL Turn\_off\_xrdy

LDMFD R13!,{R0,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@-----------------------------------------------------------------------------------@

@ PROCEDURE: Transmit status (XRDY) is generated when the CPU needs to put more data

@ in the I2C\_DATA register after the transmitted data has been shifted out on the SDA

@ pin. This procedure: Wait the bit XRDY before transmitting the next data byte

**Wait\_transmit:**

STMFD R13!,{R0-R1,R14} @ Store uses registers on Stack

@ Wait xrdy for transmitting

**Wait\_xrdy:**

LDR R0,=0x4802A024 @ IRQ STATUS RAW

LDR R1,[R0]

TST R1,#0x10 @ Test Bit 4

BEQ Wait\_xrdy

LDMFD R13!,{R0-R1,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@ ----------------------------------------------------------------------------------@

@ PROCEDURE: Wait\_access

@ Registers-ready-for-access (ARDY) is generated by the I2C when the previously

@ programmed address, data, and command have been performed and the status bits have

@ been updated. This bit is used to let the CPU know that the I2C registers are ready

@ for access. This procedure: Wait the bit ARDY before transmitting the next packet

**Wait\_access:**

STMFD R13!,{R0-R1,R14} @ Store uses registers on Stack

@ Wait xardy for accessing

**Wait\_ardy:**

LDR R0,=0x4802A024 @ IRQ STATUS RAW

LDR R1,[R0]

TST R1,#0x04 @ Test Bit 2

BEQ Wait\_ardy

LDMFD R13!,{R0-R1,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@-----------------------------------------------------------------------------------@

@ PROCEDURE: Turn\_off\_xrdy

@ The CPU poll this bit to write the next transmitted data into the I2C\_DATA register

@ This procedure: poll the bit XRDY

**Turn\_off\_xrdy:**

STMFD R13!,{R0-R1,R14} @ Store uses registers on Stack

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

MOV R1,#0x10 @ Write 1 to clear xrdy

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@-----------------------------------------------------------------------------------@

PROCEDURE: Turn\_off\_ardy

@ The CPU poll this bit for stopping access

@ This procedure: poll the bit ARDY

**Turn\_off\_ardy:**

STMFD R13!,{R0-R1,R14} @ Store uses registers on Stack

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

MOV R1,#0x4 @ Write 1 to clear ardy

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@-----------------------------------------------------------------------------------@

@ PROCEDURE: Delay 10ms

**Delay:**

STMFD R13!,{R0-R1,R14} @ Store uses registers on Stack

LDR R0,=0xFFFF @ Delay 10ms

**wait\_delay:**

SUBS R0,R0,#1

BNE wait\_delay

LDMFD R13!,{R0-R1,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@-----------------------------------------------------------------------------------@

**.align** 2

**SYS\_IRQ:** .WORD 0x0 @ Location to store systems IRQ address

**.data**

@ Data for Duty cycle PWM

**value:** **.word** 0x186, 0x30D, 0x61A, 0xC35, 0x186A, 0x30D4, 0x61A8, 0x927C, 0xAAE6, 0xB71B

@ Data for value clock 00:00:00

@ Unit and tenth of Second

**unit\_second:** **.word** 0x30

**tenth\_second:** **.word** 0x30

@ Unit and tenth of Minute

**unit\_minute:** **.word** 0x30

**tenth\_minute:** **.word** 0x30

@ Unit and tenth of Hour

**unit\_hour:** **.word** 0x30

**tenth\_hour:** **.word** 0x30

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

.END