High level algorithm

```
Wait
      if button interrupt (while in wait loop)
            check if R12 is 0 or 1 (run wait loop vs run motor)
            if R12 == 0 (last state was wait loop)
                  Turn off interrupt
                  Set bool R12 to 1 (to run motor)
                  Return to wait loop
            if R12 == 1 (last state was motor loop)
                  Turn off interrupt
                  Set R12 to 0 (wait in the wait loop)
                  Return to wait loop

      if bool R12 == 1 (while in wait loop)
            Transmit word to address
            if address array index == last address
                  Reload address array
            else
                  Increment address array index
            if word array index == last word
                  Reload word array
            else
                  increment word array index
      else
            Return to wait loop
```

Low level algorithm

```
Setup stacks
Load pin address and motor drive word arrays
Set counters for the num of addresses (16) and the num of steps (16
addresses * 50 steps = 800)
Set up counters for delay loops
Set bool value R12 to 0, so the program waits in the wait loop at start-up

Initialize I2C pins
Set up prescaler, SCLL, SCLH, OA, CON, SA and CNT registers for I2C

Reset CON register, poll bus busy, set START/STOP for CON register, poll
transmit ready
Write setup word for mode 1 to motor driver
```

```
Delay 4000 cycles

Reset CON register, poll bus busy, set START/STOP for CON register, poll
transmit ready
Write setup word for mode 2 to motor driver
Delay 4000 cycles

Reset CON register, poll bus busy, set START/STOP for CON register, poll
transmit ready
Write setup word for prescaler to motor driver
Delay 4000 cycles

Initialize clock
Enable GPIO1_8 button debounce
Setup GPIO1_8 button as input
Initialize interrupt controller

wait
      if button interrupt
            check if R12 is 0 or 1 (run wait loop vs run motor)
            if R12 == 0 (last state was wait loop)
                  Turn off interrupt
                  Set bool R12 to 1 (to run motor)
                  Return to wait loop
            if R12 == 1 (last state was motor loop)
                  Turn off interrupt
                  Set R12 to 0 (wait in the wait loop)
                  Return to wait loop

            if bool R12 == 1
                  Reset CON register, poll bus busy, set START/STOP for CON
register, poll transmit ready
                  Transmit word to address
                  Delay 60000 cycles

                  if address array index == last address
                        Reload address array

                        if word array index == last word
                              Reload word array
                              if step counter == 0 (motor turned 360
degrees)
                                    Reset all counters, reload array
```

```
pointers, reload first address and word to R4 and R5
                                    Reset R12 to 0 (waiting status)
                                    Exit the interrupt and return to wait
loop
                              else
                                    Decrement step counter
                        else
                              increment word array index
                  else
                        Increment address array index
            else
                  Return to wait loop
```

General notes

PIN INITIALIZATION

Pin 20 – Mode 0 – spi0_d0 – offset 954h
Pin 21 – Mode 0 – spi0_sclk – offset 950h

Pin 20 – Mode 2 – I2C2_SCL
Pin 21 – Mode 2 – I2C2_SDA

Control Module base address – 0x44E1_0000

Word to mux set up – 0000 0000 0000 0000 0000 0000 0011 0010, 0x0000 0032

CLOCK INITIALIZATION

Clock module base address – 0x44E0_0000
CM_PER_I2C2_CLKCTRL offset – 44h
Word to CLK – 0x2

I2C INITIALIZATION

PRESCALER

Base address – 0x4819_C000
Offset – 0xB0
Word – 0x1

Page 4369

Table 21-30. I2C_PSC Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	
7-0	PSC	R/W	0h	Fast/Standard mode prescale sampling clock divider value. The core uses this 8-bit value to divide the system clock (SCLK) and generates its own internal sampling clock (ICLK) for Fast and Standard operation modes. The core logic is sampled at the clock rate of the system clock for the module divided by (PSC + 1). Value after reset is low (all 8 bits). 0h = Divide by 1 1h = Divide by 2 FFh = Divide by 256

SCLL (low)

Base address – 0x4819_C000
Offset – B4h
1/400k=0.25 us
Word to SCLL = 0.25us * 24 MHz – 7 = 60 – 7 = 53 = 0x35, 0011 0101

21.4.1.23 I2C_SCLL Register (offset = B4h) [reset = 0h]

I2C_SCLL is shown in Figure 21-38 and described in Table 21-31.

CAUTION: During an active mode (I2C_EN bit in I2C_CON register is set to 1), no modification must be done in this register. Changing it may result in an unpredictable behavior. This register is used to determine the SCL low time value when master.

Figure 21-38. I2C_SCLL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SCLL							
R-0h																								R/W-0h							

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; -*n* = value after reset

Table 21-31. I2C_SCLL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	
7-0	SCLL	R/W	0h	Fast/Standard mode SCL low time. I2C master mode only, (FS). This 8-bit value is used to generate the SCL low time value (tLOW) when the peripheral is operated in master mode. tLOW = (SCLL + 7) * ICLK time period, Value after reset is low (all 8 bits).

SCLH (high)

Base address – 0x4819_C000
Offset – B8h
1/400k=0.25 us
Word to SCLH = 0.25us * 24 MHz – 5 = 60 – 5 = 55 = 0x37, 0011 0111

21.4.1.24 I2C_SCLH Register (offset = B8h) [reset = 0h]

I2C_SCLH is shown in Figure 21-39 and described in Table 21-32.

CAUTION: During an active mode (I2C_EN bit in I2C_CON register is set to 1), no modification must be done in this register. Changing it may result in an unpredictable behavior. This register is used to determine the SCL high time value when master.

Figure 21-39. I2C_SCLH Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								SCLH							
R-0h																								R/W-0h							

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; -*n* = value after reset

Table 21-32. I2C_SCLH Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	
7-0	SCLH	R/W	0h	Fast/Standard mode SCL low time. I2C master mode only, (FS). This 8-bit value is used to generate the SCL high time value (tHIGH) when the peripheral is operated in master mode. - tHIGH = (SCLH + 5) * ICLK time period. Value after reset is low (all 8 bits).

OA

Base address – 0x4819_C000
Offset – A8h
Word – 0x0

SA INITIALIZATION

Base address – 0x4819_C000
Offset – ACh
Word - 0xE0

I2C_CNT

Base address – 0x4819_C000
Offset – 98h
Word – 0x2

PCA INITIALIZATION

MODE 1

Base address – 0x4819_C000
Offset – 00h
Word – 0x81

MODE 2

Base address – 0x4819_C000
Offset – 01h
Word – 0x04

SET PRESCALER

[1111 1110 | 0000 0101]

Base address – 0x4819_C000
Offset – FEh
Word – 0x05

I2C TRANSMISSION (ALL STEPS NEED TO HAPPEN EACH TIME A TRANSMISSION OCCURS)

CONFIGURATION

Base address – 0x4819_C000
Offset – A4h
Word – 1000 0110 0000 0011, 0x8603

POLL BB IRQ

First, poll the BB bit (make sure it's 0). Then write 0x8603 word to 0x4819_C0A4 to start/stop automatically. Then proceed to transmit.

Base address – 0x4819_C000
Offset – 24h

LDR Rx_1, =0x4819_C024
MOV Rx_2, #0x1000
AND Rx_1,Rx_2,Rx_1
TEQ Rx_1, #0x00001000

POLL TRANSMIT DATA READY (IRQSTATUS_RAW)

LDR Rx_1, =0x4819_C024
MOV Rx_2, #0x10
AND Rx_1,Rx_2,Rx_1
TEQ Rx_1, #0x10

WRITE DATA

Initially, send [1111 1101 | 0001 0000] to close the main switch that allows individual PWM signals to come through.
Send [0000 1111 | 0001 0000] and [0010 0011 | 0001 0000] to set the PWMA and PWMB from the get go.

Base address – 0x4819_C000
Offset – 2Ch
Word - 0x10

Base address – 0x4819_C000
Offset – 9Ch
Word – the data
Reset after each write

Step		PWMA	AIN1	AIN2	A1	A2	PWMB	BIN1	BIN2	B1	B2
			21	22				17	16		
			PWM4	PWM3				PWM5	PWM6		
Address		0000 1111	0001 0111	0001 0011			0010 0011	0001 1011	0001 1111		
word	1	1	0001 0000	0000 0000	1	0	1	0000 0000	0001 0000	1	0
	2	1	0000 0000	0001 0000	1	0	1	0000 0000	0001 0000	0	1
	3	1	0000 0000	0001 0000	0	1	1	0001 0000	0000 0000	0	1
	4	1	0001 0000	0000 0000	0	1	1	0001 0000	0000 0000	1	0

Design log

3/14/23

Determined the I2C setup for prescaler, SCLL, SCLH, OA, SA and CNT

3/17/23

Determined the I2C transmission params (CON, BB and XRDY steps)

3/19/23

Determined the PCA initialization settings, the addresses for the pins and the words to send to make the motor go 260 degrees

3/21/23

Wrote the initial program, it didn't work

Fixed the code algorithm, introduced function calls for the transmission steps that had to happen each time a transmission occurred (CON, BB and XRDY). Was trying to use the timer instead of delay loops but reverted to delay loops as the delay had to happen during the setup as well, at which point managing the PC with the timer interrupt looked too complex.

3/23/23

Confirmed the correct signals were coming through to the PCA board.

Determined lack of delay in some transmissions that cause BB to get stuck and not allow further transmissions. Fixed.

Determined some configs were supposed to be transmitted to PCA instead of just storing words to addresses in assembly, so some settings were not coming through. Rewrote the code to transmit those settings. Fixed.

3/24/23

Determined I needed to put PCA to sleep before transmitting prescaler settings to it. Fixed.

Determined the all LEDs low was set to 1, which turned them all off, that was why the PCA as not running the motor even though individual step instructions were coming through. Reset it to 0 and the motor started working. Fixed.

The motor works in the “part 1” format, i.e. without interrupts. Introduced button interrupt, had an issue with interrupt not working. Luis said I needed to do more research.

3/30/23

Rechecked the interrupt settings, made sure to reset the interrupt at the beginning of the program, interrupt still not working correctly.

4/1/23

Made sure to clear the interrupt after each motor cycle is completed, the interrupt now works well. Adjusted the counters and the code to restart the motor loop correctly each time. The program is now fully correct.

Code

```
@ BBB communicating through I2C with motor driver
@ Transmits a series of steps to run a stepper motor 360 degrees
@ Every time a button is pressed

@ Uses R0-R12

@ Dmitrii Fotin March 2023

.text
.global _start
.global INT_DIRECTOR
_start:
      @stack initialization
      LDR R13,=STACK1
      ADD R13,R13,#0x1000
      CPS #0x12
```

```
      LDR R13,=STACK2
      ADD R13,R13,#0x1000
      CPS #0x13

      MOV R7,#5000            @delay loop counter for setup transmissions
      MOV R10,#60000          @delay loop counter between each step
transmission
      LDR R8, =PWM_Adresses   @load address array pointer
      LDR R9, =PWM_Words            @load word array pointer
      LDRH R4, [R8]           @load first address to R4
      LDRH R5, [R9]           @load first word to R4
      MOV R11,#800            @50 steps * 16 transmits per step
      MOV R6,#16              @16 transmits
      MOV R12,#0              @Status bool value (motor cycle vs wait loop)

      @initialize clock
      LDR R0,=0x44E00044
      MOV R1,#0x00000002
      STR R1,[R0]

      @initialize I2C pins
      LDR R0,=0x44E10954
      MOV R1,#0x00000032
      STR R1,[R0]

      LDR R0,=0x44E10950
      MOV R1,#0x00000032
      STR R1,[R0]

      @sleep
      LDR R0,=0x4819C010
      MOV R1,#0x00000002
      STR R1,[R0]

      @prescaler
      LDR R0,=0x4819C0B0
      MOV R1,#0x00000003
      STR R1,[R0]

      @SCLL
      LDR R0,=0x4819C0B4
      MOV R1,#0x00000008
      STR R1,[R0]
```

```
@SCLH
LDR R0,=0x4819C0B8
MOV R1,#0x0000000A
STR R1,[R0]

@OA
LDR R0,=0x4819C0A8
MOV R1,#0x00000000
STR R1,[R0]

@I2C CON
LDR R0,=0x4819C0A4
MOV R1,#0x00008600
STR R1,[R0]

@slave address
LDR R0,=0x4819C0AC
MOV R1,#0x00000070
STR R1,[R0]

@I2C_CNT
LDR R0,=0x4819C098
MOV R1,#0x00000002
STR R1,[R0]

@Initial PCA setup

@MODE 1
BL I2C_CON
BL POLL_BB
BL I2C_CON_START_STOP
BL POLL_XRDY

MOV R4,#0x00000000
MOV R5,#0x00000011
LDR R0,=0x4819C09C
STR R4,[R0]
STR R5,[R0]
BL DELAY

@PRESCALE
BL I2C_CON
```

```
	BL POLL_BB
	BL I2C_CON_START_STOP
	BL POLL_XRDY

	MOV R4,#0x000000FE
	MOV R5,#0x00000003
	LDR R0,=0x4819C09C
	STR R4,[R0]
	STR R5,[R0]
	BL DELAY

	@MODE 1
	BL I2C_CON
	BL POLL_BB
	BL I2C_CON_START_STOP
	BL POLL_XRDY

	MOV R4,#0x00000000
	MOV R5,#0x00000081
	LDR R0,=0x4819C09C
	STR R4,[R0]
	STR R5,[R0]
	BL DELAY

	@MODE 2
	BL I2C_CON
	BL POLL_BB
	BL I2C_CON_START_STOP
	BL POLL_XRDY

	MOV R4,#0x00000001
	MOV R5,#0x00000004
	LDR R0,=0x4819C09C
	STR R4,[R0]
	STR R5,[R0]
	BL DELAY

	@initialize clock
	LDR R0,=0x44E000AC
	LDR R1,=0x00040002
	STR R1,[R0]

	@Enable debounce on GPIO1_8 for 31 us
```

```
      LDR R0,=0x44E00150
      MOV R1,#0x00000008
      STR R1,[R0]
      LDR R0,=0x44E00154
      MOV R1,#0xA0
      STR R1,[R0]

      @set up the button input GPIO

      LDR R0,=0x4804C14C      @GPIO1 base address + falling edge offset
assignment to R0
      MOV R1,#0x00000008      @GPIO_3 address assignment to R1
      LDR R2,[R0]             @load GPIO1 base address + falling edge
offset to R0
      ORR R2,R2,R1            @zero out the bit for GPIO_3 and store the
final word in R0
      STR R2,[R0]             @store the address of the final word in R0
      ADD R1,R0,#0x34         @GPIO1 base address + OE assignment to R0
      STR R2,[R1]             @enable GPIO_3 request to PONTRPEND1

      @initialize INTC
      LDR R0,=0x482000E8      @address of INTC_MIR_CLEAR3
      MOV R1,#0x04            @value to unmask INTC INT 98, GPIOINT1A
      STR R1,[R0]             @write to INTC_MIR_CLEAR3

      @clear interrupt
      LDR R0,=0x48200048
      MOV R1,#0x01
      STR R1,[R0]

      MRS R3, CPSR            @copy CPR to R3
      BIC R3,#0x80            @clear bit 7
      MSR CPSR_c,R3           @write back to CPSR

      B WAIT_FOR_INTERRUPT

@Delay loop for PCA setups
DELAY:
      SUBS R7,R7,#1
      BNE DELAY
      MOV R7,#5000
      MOV PC, LR
```

```
@Delay loop between each motor step transmission
DELAY2:
      SUBS R10,R10,#1
      BNE DELAY2
      MOV R10,#60000
      MOV PC, LR

@the following 4 functions are called each time anything is to be
transmitted to PCA
I2C_CON:
      @I2C CON - reset

      LDR R0,=0x4819C0A4
      MOV R1,#0x00008600
      STR R1,[R0]
      MOV PC, LR

POLL_BB:
      @poll BB - check if I2C bus is available for transmission
      LDR R0, =0x4819C024
      LDR R1, [R0]
      MOV R2, #0x00001000
      AND R2,R2,R1
      TEQ R2, #0x00000000
      BNE POLL_BB
      MOV PC, LR

I2C_CON_START_STOP:
      @I2C CON - set START/STOP for CON register
      LDR R0,=0x4819C0A4
      MOV R1,#0x00008603
      STR R1,[R0]
      MOV PC, LR

POLL_XRDY:
      @check if PCA is ready to receive
      LDR R0, =0x4819C024
      LDR R1, [R0]
      MOV R2, #0x10
      AND R2,R2,R1
      TEQ R2, #0x10
      BNE POLL_XRDY
      MOV PC, LR
```

```
      @empty loop that waits for button interrupt, if status bool R12 is 1,
go to motor loop
WAIT_FOR_INTERRUPT:
      NOP
      TEQ R12,#0
      BNE SETUP
      B WAIT_FOR_INTERRUPT

      @chaining the interrupt vector with custom instructions
INT_DIRECTOR:
      STMFD SP!,{R0-R3,LR}          @push registers on stack
      LDR R0,=0x482000F8                  @address of INTC-PENDING-IRQ3
      LDR R1,[R0]                         @read INTC-PENDING-IRQ3
      TEQ R1,#0x00000004                  @check bit 2
      BNE WAIT_FOR_INTERRUPT        @if not from GPIOINT1A, go to wait loop
      LDR R0,=0x4804C02C                  @if yes, load GPIO1_IRQSTATUS_0
address
      LDR R1,[R0]                         @read STATUS
      TEQ R1,#0x00000008                  @check if bit 3 == 1
      BEQ INT_CLEAR                       @if yes, go to check/set bool for
LED loop vs. empty loop
      LDR R0,=0x48200048                  @address of INTC_CONTROL
      MOV R1,#0x01                        @clear bit 0
      STR R1,[R0]                         @write
      LDMFD SP!,{R0-R3,LR}          @restore registers
      SUBS PC,LR,#4                       @return to PC address

INT_CLEAR:
      MOV R1,#0x00000008                  @turn off GPIO_3 interrupt
      STR R1,[R0]                         @write to GPIO1_IRQSTATUS_0

      ADD R12,#1                          @Set status to 1 (to run motor
loop)

      LDR R0,=0x48200048                  @address of INTC_CONTROL
      MOV R1,#0x01                        @clear bit 0
      STR R1,[R0]                         @write
      LDMFD SP!,{R0-R3,LR}          @restore registers
      SUBS PC,LR,#4                       @return to PC address

SETUP:
      @close the main OFF switch
```

```
      BL I2C_CON
      BL POLL_BB
      BL I2C_CON_START_STOP
      BL POLL_XRDY
      LDR R0,=0x4819C09C
      MOV R4,#0x000000FD
      MOV R5,#0x00000000
      STR R4,[R0]
      STR R5,[R0]
      BL DELAY2

      BL I2C_CON
      BL POLL_BB
      BL I2C_CON_START_STOP
      BL POLL_XRDY
      @set PWMA to 1
      LDR R0,=0x4819C09C
      MOV R4,#0x0000000F
      MOV R5,#0x00000010
      STR R4,[R0]
      STR R5,[R0]
      BL DELAY2

      BL I2C_CON
      BL POLL_BB
      BL I2C_CON_START_STOP
      BL POLL_XRDY
      @set PWMB to 1
      LDR R0,=0x4819C09C
      MOV R4,#0x00000023
      MOV R5,#0x00000010
      STR R4,[R0]
      STR R5,[R0]
      BL DELAY2

TRANSMIT:

      @Load current PCA pin address and the word to send to it (2 bytes)
      LDRH R4,[R8]
      LDRH R5,[R9]

      BL I2C_CON
      BL POLL_BB
```

```
      BL I2C_CON_START_STOP
      BL POLL_XRDY

      @Transmit both bytes to PCA
      LDR R0,=0x4819C09C
      STR R4,[R0]
      STR R5,[R0]
      BL DELAY2

      SUB R6,R6,#1                  @decrement the address counter
      SUBS R11,R11,#1         @decrement the instruction counter
      BEQ END                       @if instruction counter is 0, i.e.
motor's run 360 degrees, go to END branch
      TEQ R4,#0x1F                  @if the last address is encountered
      BEQ RESET_ADDRESS_ARRAY @go to load the first address to R4 again
      ADD R8,R8,#2                  @else increment the address and the
word array indices by 2 bytes each
      ADD R9,R9,#2

      B TRANSMIT                    @go back to the beginning of this
branch to transmit the next index address and word

RESET_ADDRESS_ARRAY:
      LDR R8, =PWM_Adresses
      TEQ R6, #0x00                 @if the address counter is not 0, go to
increment the word array index
      BNE INC_WORD_ARRAY            @else, reset the word array index
RESET_WORD_ARRAY:
      LDR R9, =PWM_Words
      MOV R6,#16                    @reset address counter
      B TRANSMIT
INC_WORD_ARRAY:
      ADD R9, R9, #2                @increment the word array by 2 bytes
      B TRANSMIT

END:
      @reset all counters
      MOV R11,#800
      MOV R6,#16
      MOV R12,#0

      BL I2C_CON
      BL POLL_BB
```

```
      BL I2C_CON_START_STOP
      BL POLL_XRDY
      @open the main OFF switch to turn off all PCA pins
      LDR R0,=0x4819C09C
      MOV R4,#0x000000FD
      MOV R5,#0x00000001
      STR R4,[R0]
      STR R5,[R0]
      BL DELAY2

      @reload array pointers
      LDR R8,=PWM_Adresses
      LDR R9,=PWM_Words

      @clear interrupt
      LDR R0,=0x48200048
      MOV R1,#0x01
      STR R1,[R0]

      @go back to wait loop until next button press
      B WAIT_FOR_INTERRUPT

.data
PWM_Adresses: .HWORD 0x17, 0x15, 0x1B, 0x1F
PWM_Words: .HWORD 0x10, 0x00, 0x00, 0x10, 0x00, 0x10, 0x00, 0x10, 0x00,
0x10, 0x10, 0x00, 0x10, 0x00, 0x10, 0x00
.align 2
STACK1:     .rept 1024
            .word 0x0000
            .endr
STACK2:     .rept 1024
            .word 0x0000
            .endr

.END
```