30/03/2020 B829844

Task 3 – Data processing calculations on Cortex-M4 with reduced execution time report

This short report outlines the changes made to reduce execution time of given ARM assembly code from 935 cycles to 278 cycles, around 70% less cycles using original data.

Original code	Replaced with in modified code	Changes made to reduce execute time
`MOV r0,#(1+sampled_end-sampled)-(filter_end-filter)`	`MOV r0,#(1+sampled_end-sampled)-8`	Filter values are no longer stored separately in a memory area as according to application specification 3 the number of filter values are fixed.

`MOV r1,#0 ; coun`
`find_scaling_factor`
`MOV r2,#(filter_end-filter)`
`LDR r10,=filter ; the`
`MOV r7,#0 ; scal`
`scaling_loop`
`LDRB r5,[r10],#1 ; get`
`ADD r7,r7,r5 ; accu`
`SUBS r2,r2,#1 ; chec`
`BNE scaling_loop ; move`
	Removed from original code.	The code to find the scaling factor is no longer needed as it could be replaced with a constant, in this case the value 512 which is 2^9.

`MOV r3,#0`	Removed from original code.	Initializing count no longer required as we can overwrite the previous result with the first calculated value directly.

`MOV r9,r8`
`ADD r8,r8,#1`
	`SUB r9,r9,#7`	Address of next set of sampled data could be worked out by subtracting 7 (assuming there are 8 filter values) using the current address

`MOV r2,#(filter_end-filter)`
`LDR r10,=filter ; start`
	`MOV r3,#32`
	`MOV r4,#64`
	`MOV r5,#128`
		As filter values are embedded in the code, we do not need to load the address of the memory location storing the filter values. Instead MOV is used to store the filter values.

`filter_loop`
`LDRB r4,[r9],#1`
`LDRB r5,[r10],#1`
`MUL r6,r4,r5`
`ADD r3,r3,r6`
`SUBS r2,r2,#1`
`BNE filter_loop`
	`LDR r6,[r1],#4`
	
	`UBFX r7, r6, #0, #8`
	`MUL r8,r7,r3`
	
	`UBFX r7, r6, #8, #8`
	`SMLAD r8,r7,r3,r8`
	
	`UBFX r7, r6, #16, #8`
	`SMLAD r8,r7,r4,r8`
		4 bytes are loaded in one instruction using LDR and filtered out using UBFX removing the need of accessing memory for each byte. SMLAD – signed multiply accumulate long dual is used to multiply the filter value with the data value and accumulate in the same clock cycle. The filter values are located in the registers.

	`UBFX r7, r6, #24, #8`
	`SMLAD r8,r7,r5,r8`
	
	`LDR r6,[r1],#4`
	
	`UBFX r7, r6, #0, #8`
	`SMLAD r8,r7,r5,r8`
	
	`UBFX r7, r6, #8, #8`
	`SMLAD r8,r7,r4,r8`
	
	`UBFX r7, r6, #16, #8`
	`SMLAD r8,r7,r3,r8`
	
	`UBFX r7, r6, #24, #8`
	`SMLAD r8,r7,r3,r8`	

`UDIV r3,r3,r7`	`LSR r3,r3,#9`	Scaling down the result by dividing by 512 could be done by logical shift right LSR 9 as 2^9 = 512.

`ADD r1,r1,#1`
`CMP r0,r1`
	`SUBS r0,r0,#1`	To check if all the values has been processed, counting down is used to reduce 1 cycle. Flag is used in conjunction with instruction SUB to remove the need of instruction CMP for the conditional branch instruction.

`filter    ; 8-bit unsigned filter values`
`  DCB 32,32,64,128,128,64,32,32`
`filter_end`
	Removed from original code.	Filter values are embedded in the code and therefor this is no longer required.