Project 3 Report

Profiling and Optimization

Submitted by Siddhartha Kumar (skumar15)

Introduction

The objective of this project was to render the graphics for a dial and an animated needle that can move to a given position on the input value given .Our main aim was to first get the graphics rendered and working, profile the optimized code and then try to reduce the total number of ticks taken by focusing on the most time consuming operations.

Optimizations

The profiling revealed the biggest time consumers to be the SPI_Send() function , F_Mul(), Sin_Small() and F_Add(). SPI_Send() was our main time consuming factor and remained to be throughout the optimization, but was reduced significantly by reducing the requirement to communicate with the LCD display too often.

The calculation functions were the next big time consumers, and were focused on by using an approximation instead of using the library functions and also reducing the number of time an operation had to be executed.

The code was optimized to an extent to still make it a flexible code, and the use of hardcode for certain parameters for our specific use case was avoided, this could have lead to further reduction in time. A high speed optimization setting was set for the compiler to improve the effectiveness of optimization.

Time(ticks)	Optimizations
16604	This is the non-optimized code running an unfilled needle from 0 to 10.0 across the dial. This code was purely used to get the functionality. We had entire LCD beign refreshed for every change in the Needles position, causing us to redraw static background elements like the Dial. Also the default baud rate was used for the SPI communication with the LCD controller. The use of the math.h library for the ease of calculations was also included.
13039	The first optimization was the use of a quadratic approximation for the sin and cosine functions used for both calculation of the needle position and drawing of the dial. This saw a drop in the use of the Sin_small() function and the reduced use of the F_Mul() function. The distributive property was used to reduce the number of multiplications required.
9369	Next was a simple matter of increasing the frequency of the SCLK using Applilet , to a value of 800000, which increased the rate of communication with the LCD and hence drastically reduced the clock ticks spent communicating with the it.
3734	This was the most impactful optimization which was dividing the frame buffer into two components the background and foreground this allowed me to retain the background while only updating the foreground. Hence by only updating the pixels that were concerned with the foreground(needle) we eliminated a lot of write operations to the LCD controller. Though there was a trade off with the use of an extra buffer and a flag bit for each pixel that needs to be updated
1420	The drawback in the previous algorithm was that whenever a pixel was cleared from the Foreground buffer was it was cleared every cycle after that, therefore by incorporating a second check for this case and making sure once cleared a pixel is not touched unless written to again, we saw a reduction in more than half the time.
1182	As we were not updating entire rows at a time we needed to send a command every time to shift columns, hence not taking advantage of the auto increment feature of the LCD controller, hence by adding a check for if consecutive columns were being written to we were able to reduce the impact of multiple writes using the YRDKLR78_Command function to the LCD controller.
954	Till now the needle was always redrawn using the foreground buffer and pixels that might have been already colored in at the previous position of the needle were rewritten. By using a previous foreground buffer we were able to keep track of pixels that needed no updating. This didn't see much impact in the unfilled needle as there weren't too many pixels that were kept constant. But would see more advantage in the filled needle
1386	The Code was changed to implement the filled needle option which leads to more writes to the LCD controller, though the current implementation is still not perfect, with pixels outside the needle being drawn to. There wasn't too big an impact of the added overhead due to the previous optimzations.

Original Profile		Profile 2		Profile 3	
Total Ticks	16604	Total Ticks	13039	Total Ticks	9396
Total Regions	95	Total Regions	91	Total Regions	91
HWSDIV_16_16_16	0	HWSDIV_16_16_16	0	HWSDIV_16_16_16	0
HWSMOD_16_16_16	55	HWSMOD_16_16_16	37	HWSMOD_16_16_16	63
HWSDIV_32_32_32	0	HWSDIV_32_32_32	0	HWSDIV_32_32_32	0
CHGSIGN_32	0	CHGSIGN_32	0	CHGSIGN_32	0
SPI_Init	0	SPI_Init	0	SPI_Init	0
SPI_Send	7198	SPI_Send	7282	SPI_Send	4072
YRDKRL78_Comman	1	YRDKRL78_Comman	0	YRDKRL78_Comman	0
YRDKRL78_DataSe	18	YRDKRL78_DataSe	18	YRDKRL78_DataSe	24
FBG_InitFB	26	FBG_InitFB	27	FBG_InitFB	25
FBG_RefreshLCD	21	FBG_RefreshLCD	15	FBG_RefreshLCD	27
FBG_SetPixel	344	FBG_SetPixel	305	FBG_SetPixel	335
R_CSI21_Send_Re	678	R_CSI21_Send_Re	636	R_CSI21_Send_Re	45
LCDDrawDialNeed	37	LCDDrawDialBack	466	LCDDrawDialBack	459
TestMyCode	316	LCDDrawDialNeed	40	LCDDrawDialNeed	42
?F_ADD	2335	?F_ADD	1081	?F_ADD	1004
?FCMP_EQ	55	?FCMP_EQ	49	?FCMP_EQ	61
?FCMP_LT	136	?FCMP_LT	107	?FCMP_LT	114
?F_F2SL	380	?F_F2SL	141	?F_F2SL	130
?F_SL2F	8	?F_SL2F	17	?F_SL2F	17
?F_UL2F	362	?F_UL2F	139	?F_UL2F	150
?F_MUL	2552	?F_MUL	1477	?F_MUL	1486
fabs	93	?I_LSH_L02	467	?I_LSH_L02	360
?I_LSH_L02	486	?SI_CMP_L02	22	?SI_CMP_L02	23
?SI_CMP_L02	17	?0SFUNC_LEAVE_L	71	?0SFUNC_LEAVE_L	34
?F_NEG_L04	37	?0EFUNC_LEAVE_L	131	?0EFUNC_LEAVE_L	13
?0SFUNC_LEAVE_L	76	?0EMOVE_LONG_L0	511	?0EMOVE_LONG_L0	885
?0EFUNC_LEAVE_L	144				
?0EMOVE_LONG_L0	515				
__iar_Sin_small	605				
__iar_Dtest	109				

Profile 4		Profile Unfilled		Profile Filled	
Total Ticks	**3734**	**Total Ticks**	**954**	**Total Ticks**	**1386**
Total Regions	**93**	**Total Regions**	**93**	**Total Regions**	**94**
HWSMOD_16_16_16	2	**HWSMOD_16_16_16**	2	**HWSMOD_16_16_16**	2
SPI_Send	2484	**SPI_Send**	352	**SPI_Send**	481
YRDKRL78_Comman	18	**YRDKRL78_Comman**	1	**YRDKRL78_Comman**	3
YRDKRL78_DataSe	13	**YRDKRL78_DataSe**	6	**YRDKRL78_DataSe**	2
FBG_RefreshLCD_BG	1	**FBG_RefreshLCD_BG**	2	**FBG_RefreshLCD_BG**	5
FBG_RefreshLCD_FG	178	**FBG_RefreshLCD_FG**	244	**FBG_RefreshLCD_FG**	246
FBG_SetPixel	49	**FBG_SetPixel**	53	**FBG_SetPixel**	166
R_CSI21_Send_Re	87	**R_CSI21_Send_Re**	40	**R_CSI21_Send_Re**	59
LCDDrawDialBack	3	**LCDDrawDialBack**	4	**LCDDrawDialBack**	5
LCDDrawDialNeed	45	**LCDDrawDialNeed**	43	**DrawNeedleFilled**	90
TestMyCode	1	**TestMyCode**	0	**LCDDrawDialNeed**	44
?F_ADD	40	**?F_ADD**	40	**?F_ADD**	32
?FCMP_EQ	2	**?FCMP_EQ**	4	**?FCMP_EQ**	2
?FCMP_LT	4	**?FCMP_LT**	2	**?FCMP_LT**	3
?F_F2SL	5	**?F_F2SL**	3	**?F_F2SL**	3
?F_UL2F	1	**?F_UL2F**	3	**?F_UL2F**	0
?F_MUL	38	**?F_MUL**	31	**?F_MUL**	47
?I_LSH_L02	232	**?I_LSH_L02**	28	**?I_LSH_L02**	49
?SI_CMP_L02	12	**?SI_CMP_L02**	10	**?SI_CMP_L02**	42
?0SFUNC_LEAVE_L	43	**?0SFUNC_LEAVE_L**	9	**?0SFUNC_LEAVE_L**	12
?0EFUNC_LEAVE_L	47	**?0EFUNC_LEAVE_L**	9	**?0EFUNC_LEAVE_L**	10
?0EMOVE_LONG_L0	429	**?0EMOVE_LONG_L0**	70	**?0EMOVE_LONG_L0**	83