Stony Brook University

ESE 566: HARDWARE-SOFTWARE CODESIGN OF EMBEDDED SYSTEMS

PROJECT 1: <u>SOUND-CONTROLLED STOPWATCH</u>

GUIDED BY:
Dr. ALEX DOBOLI

PREPARED BY:

AISHWARYA GANDHI (111424452)
KEWAL RAUL (111688087)
SHASHANK RAO (111471609)

1. INTRODUCTION

The aim of the project is to build a sound-controlled stopwatch using the provided PSoC Eval1 kit. The stopwatch uses a microphone which starts and stops the timer on consecutive whistle blows. The system also contains multiple other modes which can be switched to by long presses of the button on the PSoC. One mode is that of a traditional stopwatch, which can be turned on and off using a push button. There are also additional modes - accuray mode, which can set the accuracy of the stopwatch (resolutions of 1 second, ½ seconds and 1/10 seconds); history mode, which can store the last ‘x’ timer readings; and microphone sensitivity mode, which allows the user to set the trigger level for the microphone in use. This report provides the complete summary of the project system design, the overall algorithm, the different hardware resources used in the design, as well as computational complexity of the developed routines in the system.

2. FUNCTIONAL OVERVIEW

The overall system design consists of multiple modes which must be cycled through, and then, within each mode, particular functionalities must be implemented. As per the requirements, there are five main modes which are to be implemented. These modes are as follows:

- **Push button stopwatch mode** - This mode will enable the system to operate as a traditional push button stopwatch. One short press of the button will start the stopwatch timer and another short press of the button will stop the stopwatch timer. Subsequent short presses of the button will take multiple stopwatch readings, ie, on third short press of the button, the timer will restart and collect the next reading. The stopwatch time during runtime will be displayed on the LCD depending on the resolution set in the Accuracy mode. Each timer reading taken will be stored in the memory, which can be viewed in the memory mode. Long press of the button will exit this mode and switch to the next mode.
- **Accuracy mode** - This mode allows the user to set the resolution with which he/she would like to see the time displayed during the stopwatch operation. Consequent short presses within this mode allow the user to cycle through the available resolutions - 1 second, ½ seconds and 1/10 seconds. If the resolution is set to 1 second, the time will be displayed as HH:MM:SS with increments of 1 second; if the resolution is set to ½ second, the time will be displayed as HH:MM:SS.m with increments of ½ seconds (m here has the value 0 or 5 changing every ½ second); if the resolution is set to 1/10 second, the time will be displayed as HH:MM:SS.mm with increments of 1/10 seconds (mm will range between 1-10 changing every 1/10 second). Long press of the button will exit this mode and switch to the next mode.

- **Memory mode** - This mode allows the user to view the last 'x' stopwatch readings taken. Consequent short presses will cycle through the stored readings as well as the shortest, longest and average among the readings taken. For this project we have considered **'x' = 5**, ie, last 5 readings can be saved (any subsequent readings will overwrite earlier saved readings starting with the earliest reading taken). Shortest, longest and average of the 5 saved readings (or less, if fewer readings have been taken) will also be calculated and displayed during consecutive short button presses. Long press of the button will exit this mode and switch to the next mode.
- **Sound sensitivity mode** - This mode allows the user to set the threshold over which consecutive whistle sounds will be recognized by the microphone for the operation of the sound-based stopwatch. The background noise present while in this mode will be set as the threshold; any sound greater than this set value will result in start/stop of the stopwatch in sound controlled stopwatch mode. Long press of the button will exit this mode and switch to the next mode.
- **Sound controlled stopwatch mode** - This mode enables the sound-based stopwatch mode which the user can operate. Consecutive whistles by the user will start/stop the stopwatch timer. The time displayed is according to the resolution set in the Accuracy mode. 'x' readings taken will also be stored in memory to be displayed in Memory mode. Subsequent whistle blows by the user will take multiple stopwatch readings. The whistle sounds by the user should be greater than the threshold noise set in the sound sensitivity mode. Long press of the button will exit this mode and switch to the next mode.

The following diagram gives the overall functional operations of the designed system:

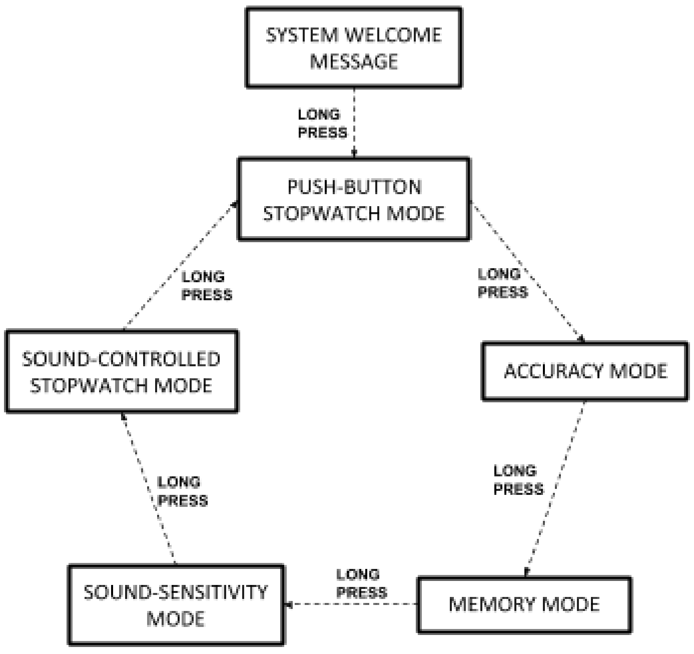


Fig. 1: Overall states of all five modes

The following figures show the control flow implemented in each mode:

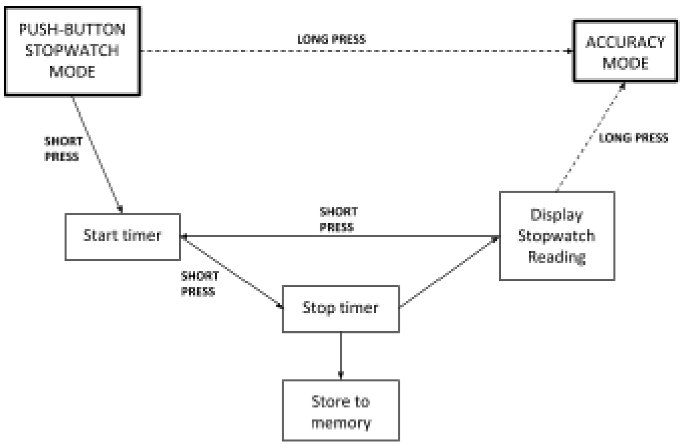


Fig. 2: Push-button stopwatch mode control flow diagram

As can be seen in Fig.2, a short press on the button, after navigating to the push button stopwatch mode, starts the stopwatch timer. Another short press of the button results in stopping the stopwatch timer and the collected reading is stored to memory while being displayed on the LCD at the same time. Another short press of the button restarts the timer, while a long press of the button ends the push-button mode and enters the Accuracy mode.

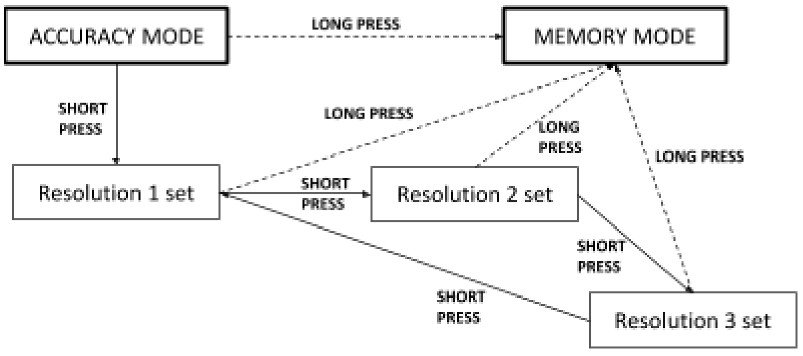


Fig. 3: Accuracy mode control flow diagram

As depicted in Fig.3, a short press of the button sets 1 second (Resolution 1) as the resolution. If a long press is made now, this 1 second resolution is set for the timers in both the push-button and sound-based stopwatch. If another short press is made, resolution of ½ seconds (Resolution 2) is set, if long press of button is made next. For a third short button press, the resolution of 1/10 seconds (Resolution 3) is set. Further short presses of the button result in cyclic setting of these three resolutions. A long press after any resolution is set, causes the control flow to exit of Accuracy mode and enter Memory mode.

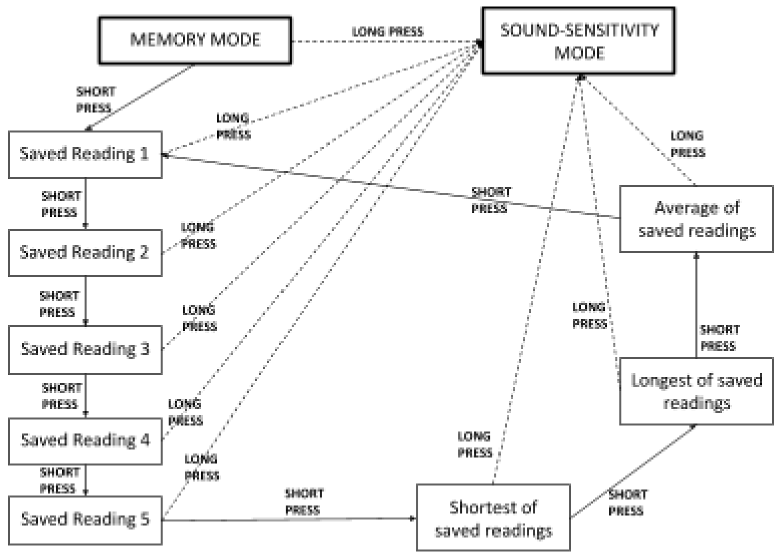


Fig. 4: Memory mode control flow diagram

Fig.4 shows Memory mode functional operation of the stopwatch. Short button presses causes the control flow to display the last 5 saved readings along with their averages, shortest and longest times. Long pressing the button during display of any of the saved readings causes the end of Memory mode and start of the Sound-sensitivity mode.

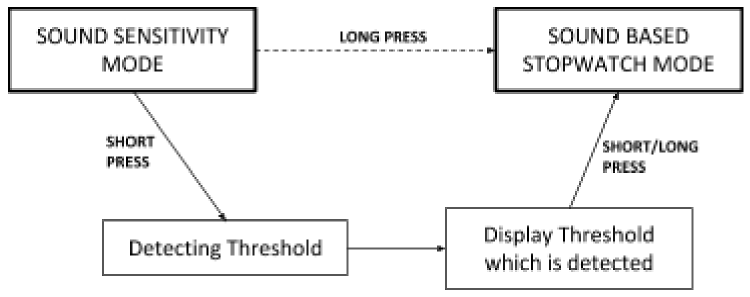


Fig. 5: Sound sensitivity mode control flow diagram

Fig. 5 shows control flow diagram for Sound-sensitivity mode. Short press of the button starts the sensor which detects the current noise levels present in the current environment and sets the threshold as twice noise level detected. This means that when in sound-based stopwatch mode, two times of the noise detected in the current environment is taken as the trigger for start/stop operations. This mode will simply display the calculated threshold and wait for a short/long press. Either button press at this moment will exit this mode and enter the last mode, which is Sound-based stopwatch mode.

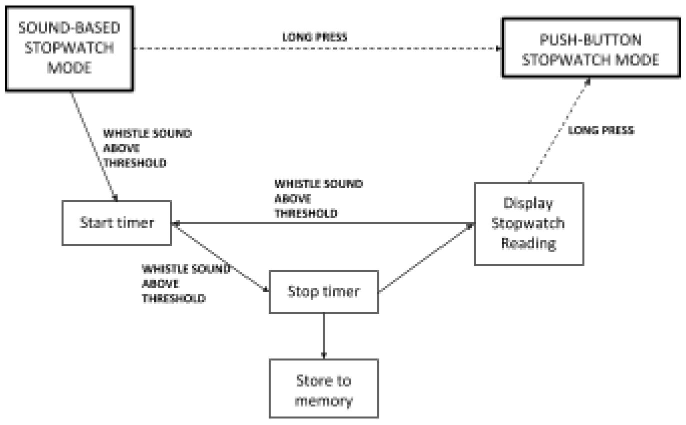

SOUND-BASED STOPWATCH MODE
LONG PRESS
PUSH-BUTTON STOPWATCH MODE
WHISTLE SOUND ABOVE THRESHOLD
LONG PRESS
WHISTLE SOUND ABOVE THRESHOLD
Start timer
Display Stopwatch Reading
WHISTLE SOUND ABOVE THRESHOLD
Stop timer
Store to memory

Fig. 6: Sound based stopwatch mode control flow diagram

Fig. 6 shows operation of Sound-based stopwatch mode. This operation is similar to the functioning of Push-button stopwatch mode. The only difference is the triggering mechanism. Here, in addition to push-button, whistle sounds greater than the threshold trigger can be used to start and stop the stopwatch timer. Remaining functionalities remain similar to the push-button based stopwatch.

3. TECHNICAL IMPLEMENTATION

3.1 OVERALL STRUCTURE OF PROJECT

Hardware resources used in system: The system makes use of the following hardware resources on the PSoC Eval1 kit:

- The following are some of the important Global Resources set for the entire system -
  - Vcc / SysClk => 5.0V / 24Mhz
  - CPU_Clock => SysClk/8 , i.e., 3MHz is selected as the frequency with which to run the system; this particular value was selected since the timers required to be implemented for the stopwatch required frequencies which were easier to obtain from this CPU_Clock.
  - VC1=SysClk/N => 16 ; this value was selected since (25/16 = 1.5 MHz) was required to run the ADC for sensing sound input for the microphone.
  - VC2=VC1/N => 15 ; this value was selected since (1.5/15 = 0.1MHz = 100KHz) was required for running three Timer blocks for implementing each resolution of the stopwatch.
- Push button switch on the PSoC is connected to Port 1, bit 0 (P1[0]); this button is connected as an GPIO interrupt, meaning that any time button is pressed an interrupt occurs in the system; P1[0] values set in the Interconnect tab of the PSoC Designer are -

    - Select => StdCPU
    - Drive => Pull Down
    - Interrupt => RisingEdge
- ❏ 4 LEDs on the PSoC are connected to Port 1, bits 1-4 (P1[1], P1[2], P1[3], P1[4]); a combination of these LEDs are used to display the current mode in which the system is running. Values set in the Interconnect tab of the PSoC Designer for these ports are -
    - Select => StdCPU
    - Drive => Strong
    - Interrupt => DisableInt
- ❏ A Character LCD has been used to display most of the output of the entire system. In this project, LCD has been connected to the port 2, bits 0-6 (P2[0], P2[1], P2[2], P2[3], P2[4], P2[5], P2[6]).
- ❏ 2 16-bit Timers and 1 32-bit Timer are used in order to run the timers for the three different resolutions required for the stopwatch. The following are the values set in the Interconnect tab of PSoC Designer -

	32-bit Timer for 1 second resolution (named Timer32_1 in the project)	**16-bit Timer for ½ second resolution (named Timer16_1 in the project)**	**16-bit Timer for 1/10 second resolution (named Timer16_2 in the project)**
Clock	VC2	VC2	VC2
Capture	Low	Low	Low
TerminalCountOut	None	None	None
CompareOut	None	None	None
Period	99999	49999	9999
CompareValue	0	0	0
CompareType	Less Than Or Equal	Less Than Or Equal	Less Than Or Equal
InterruptType	Terminal Count	Terminal Count	Terminal Count
ClockSync	Sync to SysClk	Sync to SysClk	Sync to SysClk
TC_PulseWidth	Full Clock	Full Clock	Full Clock
InvertCapture	Normal	Normal	Normal

Table 1: Values to be set for the three Timers used in Project

As can be seen in Table 1, the only difference in the values set are for the Period field. These values were selected by using the equation (obtained from the datasheet for Timers) -

*OutputPeriod = SourceClockPeriod * (PeriodRegisterValue + 1)*

For an OutputPeriod of 1 second (resolution of 1 second) and SourceClockPeriod of ($VC2 = 0.1\,MHz = 10^{-5} seconds$), PeriodRegisterValue comes out to be 99999 which is set in Period field. Similarly, for an OutputPeriod of 1/2 second (resolution of ½ second) and SourceClockPeriod of ($VC2 = 0.1\,MHz = 10^{-5} seconds$), PeriodRegisterValue comes out to be 49999 and for an OutputPeriod of 1/10 second (resolution of 1/10 second) and SourceClockPeriod of ($VC2 = 0.1\,MHz = 10^{-5} seconds$), PeriodRegisterValue comes out to be 9999. These values signify that whenever the Timers countdown from the respective periods to 0, an interrupt is generated which can be captured and used to calculate time in ISR in software.

❏ The following circuit is constructed in order to implement microphone functionality:

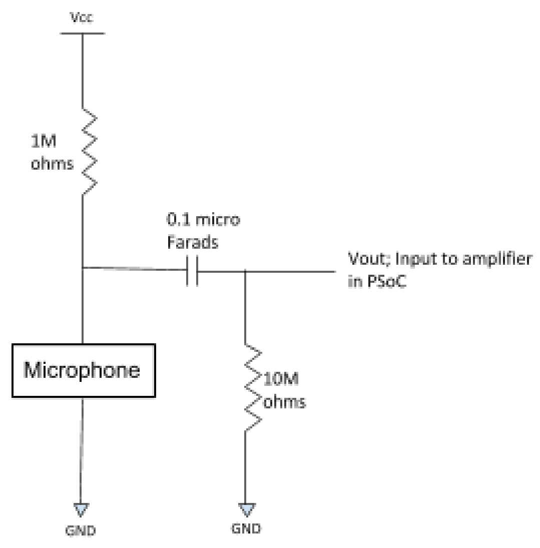


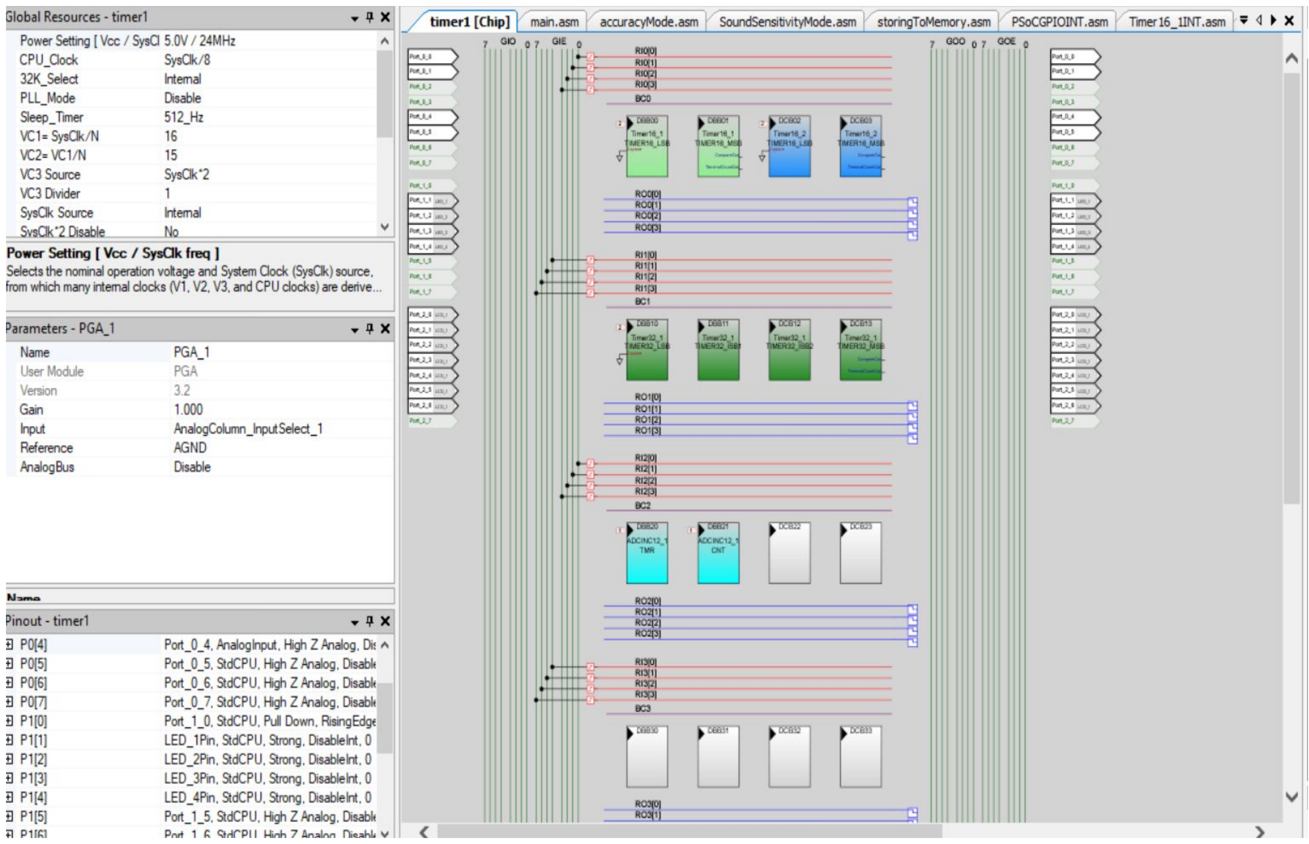
Fig. 7: Microphone Sensor circuit

The circuit shown in Fig. 7 is used to obtain input from the microphone sensor. The output Vout is passed as input to the Programmable Gain Amplifier (PGA) which is used to amplify the gain obtained from the sensor. Port 0, bit 4 (P0[4]) is used to pass Vout to PGA. Some of the values set in the Interconnect tab of PSoC Designer for this port are -

- Select => AnalogInput
- Drive => High Z Analog
- Interrupt => DisableInt

- ❑ A Programmable Gain Amplifier is used which captures the signal arriving from the microphone circuit and passes the output to the Analog-to-Digital Converter (ADC). The values set in the Interconnect tab of PSoC Designer are -
    - ○ Gain => 1.000
    - ○ Input => AnalogColumn_InputSelect_1
    - ○ Reference => AGND
    - ○ AnalogBus => Disable
- ❑ A 12-Bit Incremental ADC is also used to convert the microphone sensor data into digital form so it can be processed. The values set in the Interconnect tab of PSoC Designer are -
    - ○ ClockPhase => Norm
    - ○ TMR Clock => VC1
    - ○ Input => ACB01
    - ○ CNT Clock => VC1

The following figures (8 & 9) show the complete connections made in the Interconnection tab of PSoC Designer for the entire system:


timer1 [Chip]
main.asm
accuracyMode.asm
SoundSensitivityMode.asm
storingToMemory.asm
PSoCGPIOINT.asm
Timer16_1INT.asm
DBB00 Timer16_1 TIMER16_LSB
DBB01 Timer16_1 TIMER16_MSB
DCB02 Timer16_2 TIMER16_LSB
DCB03 Timer16_2 TIMER16_MSB
DBB10 Timer32_1 TIMER32_LSB
DBB11 Timer32_1 TIMER32_ISB
DCB12 Timer32_1 TIMER32_ISB2
DCB13 Timer32_1 TIMER32_MSB
DBB20 ADCINC12_1 TMR
DBB21 ADCINC12_1 CNT
DCB22
DCB23
DBB30
DBB31
DCB32
DCB33

Global Resources - timer1

Power Setting [Vcc / SysCl	5.0V / 24MHz
CPU_Clock	SysClk/8
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	16
VC2= VC1/N	15
VC3 Source	SysClk*2
VC3 Divider	1
SysClk Source	Internal
SysClk*2 Disable	No

Power Setting [Vcc / SysClk freq]
Selects the nominal operation voltage and System Clock (SysClk) source, from which many internal clocks (V1, V2, V3, and CPU clocks) are derive...

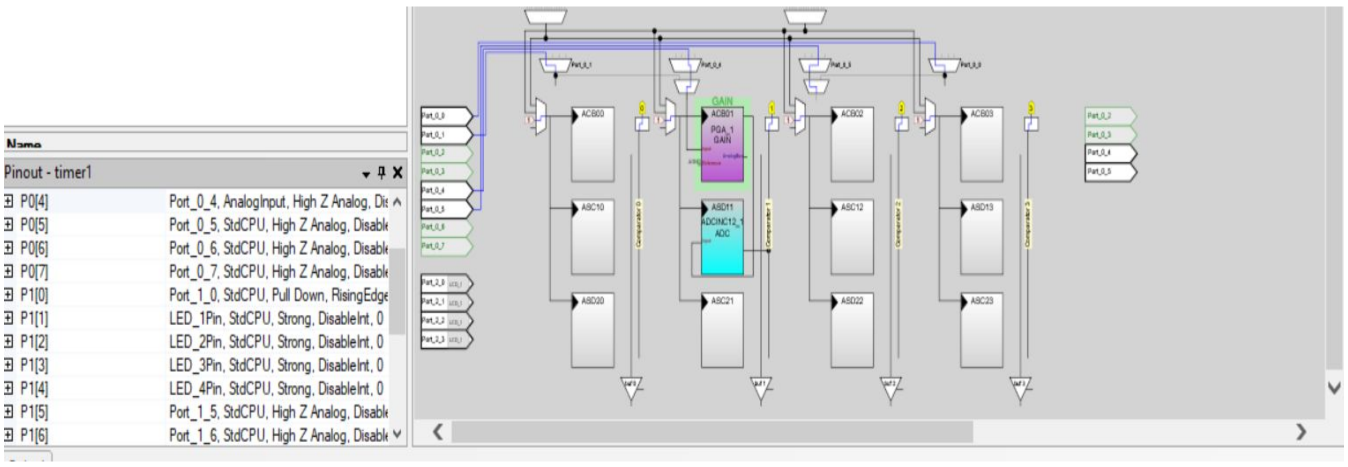
Parameters - PGA_1

Name	PGA_1
User Module	PGA
Version	3.2
Gain	1.000
Input	AnalogColumn_InputSelect_1
Reference	AGND
AnalogBus	Disable

Pinout - timer1

P0[4]	Port_0_4, AnalogInput, High Z Analog, Di
P0[5]	Port_0_5, StdCPU, High Z Analog, Disabl
P0[6]	Port_0_6, StdCPU, High Z Analog, Disabl
P0[7]	Port_0_7, StdCPU, High Z Analog, Disabl
P1[0]	Port_1_0, StdCPU, Pull Down, RisingEdg
P1[1]	LED_1Pin, StdCPU, Strong, DisableInt, 0
P1[2]	LED_2Pin, StdCPU, Strong, DisableInt, 0
P1[3]	LED_3Pin, StdCPU, Strong, DisableInt, 0
P1[4]	LED_4Pin, StdCPU, Strong, DisableInt, 0
P1[5]	Port_1_5, StdCPU, High Z Analog, Disabl
P1[6]	Port_1_6, StdCPU, High Z Analog, Disabl

Fig. 8: Implementation of Digital blocks in the PSoC Project



Pinout - timer1

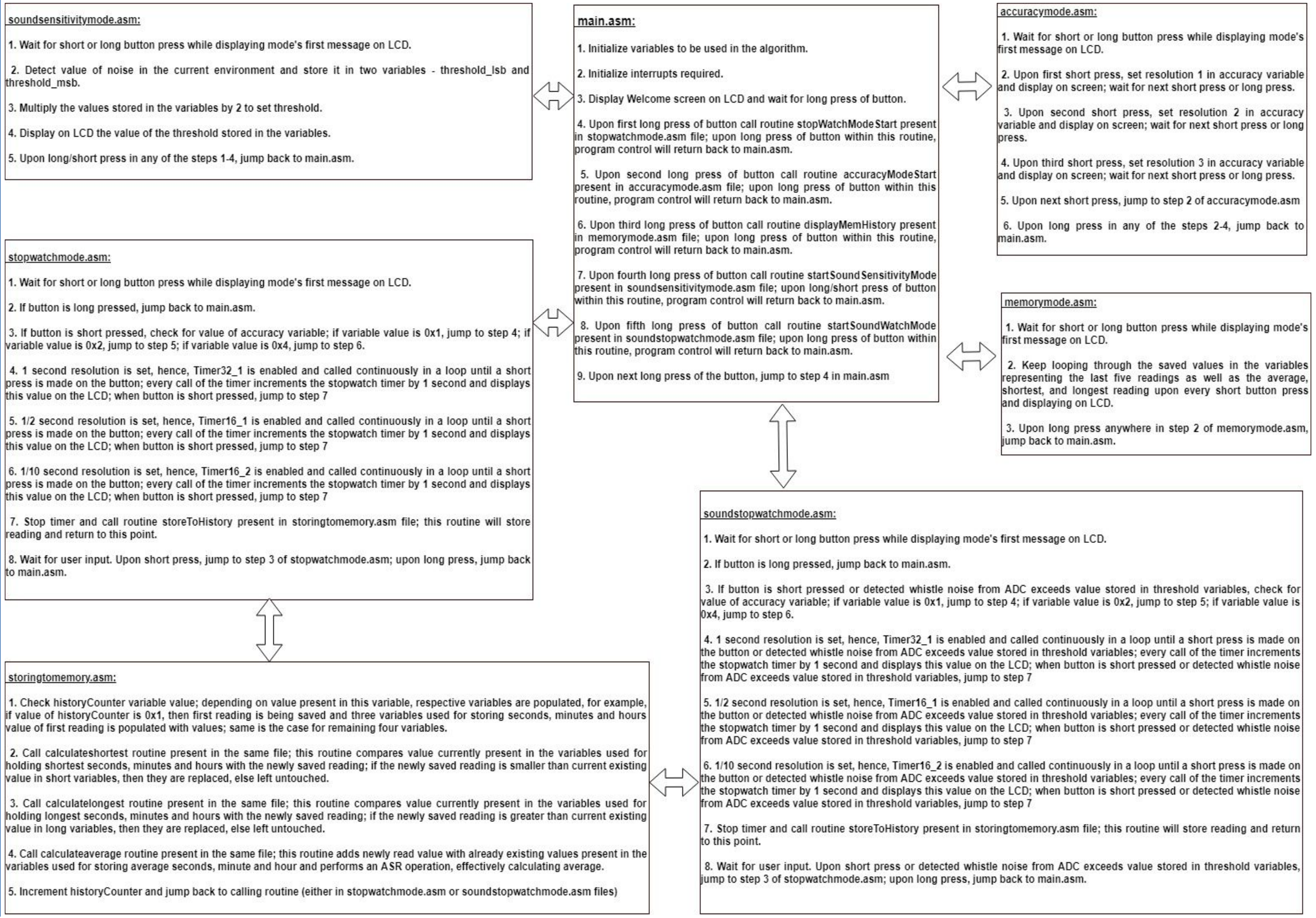
P0[4]	Port_0_4, AnalogInput, High Z Analog, Dis
P0[5]	Port_0_5, StdCPU, High Z Analog, Disabl
P0[6]	Port_0_6, StdCPU, High Z Analog, Disabl
P0[7]	Port_0_7, StdCPU, High Z Analog, Disabl
P1[0]	Port_1_0, StdCPU, Pull Down, RisingEdge
P1[1]	LED_1Pin, StdCPU, Strong, DisableInt, 0
P1[2]	LED_2Pin, StdCPU, Strong, DisableInt, 0
P1[3]	LED_3Pin, StdCPU, Strong, DisableInt, 0
P1[4]	LED_4Pin, StdCPU, Strong, DisableInt, 0
P1[5]	Port_1_5, StdCPU, High Z Analog, Disabl
P1[6]	Port_1_6, StdCPU, High Z Analog, Disabl

Fig. 9: Implementation of Analog blocks in the PSoC Project

__Software and Algorithm used in system:__ The project required creation/modification of the following files in order to implement functionality:

1. main.asm
2. delay.asm
3. stopwatchmode.asm
4. storingtomemory.asm
5. accuracymode.asm
6. memorymode.asm
7. soundsensitivitymode.asm
8. soundstopwatchmode.asm
9. PSoCGPIOINT.asm
10. Timer16_1INT.asm
11. Timer16_2INT.asm
12. Timer32_1INT.asm

The flowchart on the next page gives a brief summary of the overall algorithm on the basis of the different files which are present.



main.asm:

1. Initialize variables to be used in the algorithm.

2. Initialize interrupts required.

3. Display Welcome screen on LCD and wait for long press of button.

4. Upon first long press of button call routine stopWatchModeStart present in stopwatchmode.asm file; upon long press of button within this routine, program control will return back to main.asm.

5. Upon second long press of button call routine accuracyModeStart present in accuracymode.asm file; upon long press of button within this routine, program control will return back to main.asm.

6. Upon third long press of button call routine displayMemHistory present in memorymode.asm file; upon long press of button within this routine, program control will return back to main.asm.

7. Upon fourth long press of button call routine startSoundSensitivityMode present in soundsensitivitymode.asm file; upon long/short press of button within this routine, program control will return back to main.asm.

8. Upon fifth long press of button call routine startSoundWatchMode present in soundstopwatchmode.asm file; upon long press of button within this routine, program control will return back to main.asm.

9. Upon next long press of the button, jump to step 4 in main.asm

soundsensitivitymode.asm:

1. Wait for short or long button press while displaying mode's first message on LCD.

2. Detect value of noise in the current environment and store it in two variables - threshold_lsb and threshold_msb.

3. Multiply the values stored in the variables by 2 to set threshold.

4. Display on LCD the value of the threshold stored in the variables.

5. Upon long/short press in any of the steps 1-4, jump back to main.asm.

stopwatchmode.asm:

1. Wait for short or long button press while displaying mode's first message on LCD.

2. If button is long pressed, jump back to main.asm.

3. If button is short pressed, check for value of accuracy variable; if variable value is 0x1, jump to step 4; if variable value is 0x2, jump to step 5; if variable value is 0x4, jump to step 6.

4. 1 second resolution is set, hence, Timer32_1 is enabled and called continuously in a loop until a short press is made on the button; every call of the timer increments the stopwatch timer by 1 second and displays this value on the LCD; when button is short pressed, jump to step 7

5. 1/2 second resolution is set, hence, Timer16_1 is enabled and called continuously in a loop until a short press is made on the button; every call of the timer increments the stopwatch timer by 1 second and displays this value on the LCD; when button is short pressed, jump to step 7

6. 1/10 second resolution is set, hence, Timer16_2 is enabled and called continuously in a loop until a short press is made on the button; every call of the timer increments the stopwatch timer by 1 second and displays this value on the LCD; when button is short pressed, jump to step 7

7. Stop timer and call routine storeToHistory present in storingtomemory.asm file; this routine will store reading and return to this point.

8. Wait for user input. Upon short press, jump to step 3 of stopwatchmode.asm; upon long press, jump back to main.asm.

accuracymode.asm:

1. Wait for short or long button press while displaying mode's first message on LCD.

2. Upon first short press, set resolution 1 in accuracy variable and display on screen; wait for next short press or long press.

3. Upon second short press, set resolution 2 in accuracy variable and display on screen; wait for next short press or long press.

4. Upon third short press, set resolution 3 in accuracy variable and display on screen; wait for next short press or long press.

5. Upon next short press, jump to step 2 of accuracymode.asm

6. Upon long press in any of the steps 2-4, jump back to main.asm.

memorymode.asm:

1. Wait for short or long button press while displaying mode's first message on LCD.

2. Keep looping through the saved values in the variables representing the last five readings as well as the average, shortest, and longest reading upon every short button press and displaying on LCD.

3. Upon long press anywhere in step 2 of memorymode.asm, jump back to main.asm.

soundstopwatchmode.asm:

1. Wait for short or long button press while displaying mode's first message on LCD.

2. If button is long pressed, jump back to main.asm.

3. If button is short pressed or detected whistle noise from ADC exceeds value stored in threshold variables, check for value of accuracy variable; if variable value is 0x1, jump to step 4; if variable value is 0x2, jump to step 5; if variable value is 0x4, jump to step 6.

4. 1 second resolution is set, hence, Timer32_1 is enabled and called continuously in a loop until a short press is made on the button or detected whistle noise from ADC exceeds value stored in threshold variables; every call of the timer increments the stopwatch timer by 1 second and displays this value on the LCD; when button is short pressed or detected whistle noise from ADC exceeds value stored in threshold variables, jump to step 7

5. 1/2 second resolution is set, hence, Timer16_1 is enabled and called continuously in a loop until a short press is made on the button or detected whistle noise from ADC exceeds value stored in threshold variables; every call of the timer increments the stopwatch timer by 1 second and displays this value on the LCD; when button is short pressed or detected whistle noise from ADC exceeds value stored in threshold variables, jump to step 7

6. 1/10 second resolution is set, hence, Timer16_2 is enabled and called continuously in a loop until a short press is made on the button or detected whistle noise from ADC exceeds value stored in threshold variables; every call of the timer increments the stopwatch timer by 1 second and displays this value on the LCD; when button is short pressed or detected whistle noise from ADC exceeds value stored in threshold variables, jump to step 7

7. Stop timer and call routine storeToHistory present in storingtomemory.asm file; this routine will store reading and return to this point.

8. Wait for user input. Upon short press or detected whistle noise from ADC exceeds value stored in threshold variables, jump to step 3 of stopwatchmode.asm; upon long press, jump back to main.asm.

storingtomemory.asm:

1. Check historyCounter variable value; depending on value present in this variable, respective variables are populated, for example, if value of historyCounter is 0x1, then first reading is being saved and three variables used for storing seconds, minutes and hours value of first reading is populated with values; same is the case for remaining four variables.

2. Call calculateshortest routine present in the same file; this routine compares value currently present in the variables used for holding shortest seconds, minutes and hours with the newly saved reading; if the newly saved reading is smaller than current existing value in short variables, then they are replaced, else left untouched.

3. Call calculatelongest routine present in the same file; this routine compares value currently present in the variables used for holding longest seconds, minutes and hours with the newly saved reading; if the newly saved reading is greater than current existing value in long variables, then they are replaced, else left untouched.

4. Call calculateaverage routine present in the same file; this routine adds newly read value with already existing values present in the variables used for storing average seconds, minute and hour and performs an ASR operation, effectively calculating average.

5. Increment historyCounter and jump back to calling routine (either in stopwatchmode.asm or soundstopwatchmode.asm files)

Initially, in the main.asm file, multiple variables are defined and initialized. These variables are used for multiple purposes - such as for calculating time for stopwatch timer, as well as for storing last 5 stopwatch readings along with the shortest, longest and average of the saved readings. Each of these variables take up 1 byte of the SRAM. Five variables are used during calculation of current time within the timer interrupts - msec, milisec, sec, min and hour. 24 variables are used for storing the saved readings - mem_sec1, mem_sec2, mem_sec3, mem_sec4 and mem_sec5 are used for storing seconds value of the five saved readings; mem_min1, mem_min2, mem_min3, mem_min4, mem_min5 are used for storing minutes value of the five saved readings; mem_hour1, mem_hour2, mem_hour3, mem_hour4, mem_hour5 are used for storing hours value of the five saved readings; mem_shortsec, mem_longsec, mem_avgsec are used for storing the seconds value of the shortest reading, longest reading and average of readings among the stored readings; mem_shortmin, mem_longmin, mem_avgmin are used for storing the minutes value of the shortest reading, longest reading and average of readings among the stored readings; mem_shorthour, mem_longhour, mem_avghour are used for storing the hours value of the shortest reading, longest reading and average of readings among the stored readings - these variables are simply read and displayed during consecutive short presses of button. A separate variable historyCounter (also 1 byte of SRAM) is used to maintain the number of saved readings. It is initialized with value 0x01 and keeps incrementing until 5th reading is saved; after 5th reading is saved in the respective variables, this historyCounter variable is reset to 0x01. Every time stopwatch is stopped, depending on the historyCounter variable value, respective variables in memory are updated with the stopwatch reading at the time of the stop operation. During this time, the shortest, longest and average values are also updated; shortest and longest values are calculated by comparing the existing variable values with the new incoming stopwatch reading values. Average is calculated by adding the existing variable values with the new incoming stopwatch reading values and dividing by 2. This reduces overhead calculating separately during each display operation in Memory mode display of shortest, longest and average readings.

Some of the other important variables defined and initialized in main.asm are watchSwitch and modeSwitch (again both are 1 byte each of SRAM). A combination of watchSwitch and modeSwitch variable values is used to detect whether long or short press has been performed. As mentioned earlier, the push button has been designed as an interrupt, i.e., upon press of push button (regardless of long or short), the PSoC_GPIO_ISR interrupt service routine in the PSoCGPIOINT.asm file is called. Within this ISR, a very small delay of ½ seconds occurs. After this delay, the value of push button port bit is checked. If the value is not zero, then it means that a long press was performed on the button. In this case, watchSwitch variable value is set to 0x02 and modeSwitch variable value is set to 0x01. In case after the delay, the value of push button port bit is zero, then it means that the button was pushed less than ½ second, signifying a short button press. In this case, the modeSwitch variable value is set to 0x00 and watchSwitch variable value is toggled from its current value (in case, watchSwitch value is 0x00, it is changed to 0x01, and vise-versa; in case watchSwitch variable value is 0x02, it is updated to 0x00 following which consequent presses toggles its value). These variable values are used in many areas of the program flow to check whether long press or short press was performed.

modeCounter is another variable of 1 byte which is updated every time long press is performed between change of modes to keep track of the current mode the system is in. It is initialized with 0x01 and increments with the modes, i.e., 0x01 value signifies that current mode is push-button stopwatch mode, 0x02 value signifies that current mode is Accuracy mode, 0x03 value signifies that current mode is Memory mode, and so on. After 0x05, the value is reset to 0x01.

The final variables in use are iResult (2 bytes), threshold_lsb and threshold_msb (both 1 byte each). These variables are used for operation of sound related modes. threshold_lsb and threshold_msb are both initialized with a default value. When in Sound-sensitivity mode, these variables are updated with twice the value of noise detected through the microphone. Then, during Sound-based stopwatch mode, iResult variable is used to store the detected whistle noise and compare with the threshold variable values. If iResult value is greater than value stored in threshold variables, then the stopwatch is triggered and timer starts/stops.

Timer16_1INT.asm, Timer16_2INT.asm and Timer32_1INT.asm files were modified so as to calculate and display the stopwatch timers on the LCD. For example, Timer16_1INT.asm is the ISR written for timer of resolution ½ second. This means that every half a second, an interrupt is generated and ISR is called. On every ISR call, the value in milisec variable is incremented. After this, this variable value is checked if its even or odd. If this value is odd, the msec value is set to 0x50 to be displayed on the LCD. If the value is even, then it means that one second has passed and the sec variable value is updated and displayed on the stopwatch timer on the LCD. Similarly, when sec variable reaches 0x3B (59s), the min variable value is updated to 0x01 and sec value is updated to 0x00 and displayed on the LCD. Same happens with 59 minutes. This way each value within the timer is calculated and displayed until the button is pressed for stopwatch timer to stop. Similar execution techniques have been implemented in Timer32_1INT.asm and Timer16_2INT.asm, with each modified slightly in order to display the requisite accuracy in the stopwatch timer.

3.2 I/O PINS, FLASH MEMORY AND RAM MEMORY

The following is a brief estimation of the I/O pins, flash memory and RAM memory being used by the entire implemented project:

Number of I/O pins: 15 out of 24 port pins have been used for various inputs/outputs such as push button, LEDs and LCD

Flash Memory Usage: Around 16% full; ~5100 bytes has been used by the system

RAM Memory Usage: Around 5% full; ~80 bytes has been used by the system

3.3 COMPUTATIONAL COMPLEXITY FOR MAIN ROUTINES

The following tables list the approximate number of clock cycles, computational complexity, and required program and data memory for the main routines being used in the system:

Important Routines	**Approx. number of clock cycles**	**Computational Time Complexity**	**Approx. program memory (in bytes)**	**Approx. data memory (in bytes)**
main	700	O(1)	400	40
stopWatchModeStart	500	O(N)	220	8
storeToHistory	550	O(1)	350	30
startSoundWatchMode	670	O(N)	270	12
startSoundSensitivityMode	250	O(1)	130	4
displayMemHistory	1650	O(1)	1100	35
accuracyModeStart	350	O(1)	200	3

Table 2: Statistics regarding implemented system

Computational time complexity of the entire system is O(N). This is because none of the routines have nested for loops. The routines main, startSoundSensitivityMode, displayMemHistory, storeToHistory and accuracyModeStart have loops that only wait for a user input through the push button (storeToHistory does not even have any interaction with the user; it executes the instructions sequentially and returns to the calling function). This is why we have considered these routines to have a time complexity of O(1). Once the user provides input through the push button, the code executes sequentially (there are multiple jumps, but the flow of the jumps is sequential). The routines stopWatchModeStart and startSoundWatchMode have a time complexity of O(N) since these routines are used to run to stopwatch timers which can run upto the maximum time that has been set in the system, which, in the case of our project is 99 hours: 59 minutes: 59 seconds. This means that stopwatch timer can run upto the maximum permissible time which can be considered as N. Computational space complexity of the entire system is a constant O(1). This is because there are no dynamically allocated structures during runtime; almost all the variables are initialized and stored in the bss area of the SRAM before the start of the program flow.

As can be seen in the Table 2, maximum values for clock cycles, program and data memory are all in displayMemHistory routine. This is mainly due to the large number of lcalls, cmps and movs involved in displaying the saved results onto the LCD screen.

3.4 TESTING AND DEBUGGING PROCEDURES

All the testing and debugging was carried out by outputting the values on to the LCD screen. This resulted in a fine-tuned system, with the end product requiring little to no separate testing or debugging procedures. Since most of the testing and debugging were done during the development phase, behavior of the system was able to be easily predicted as well as captured, identified and corrected easily during the later stages of development. For example, incorrect values entered in a memory location was directly output onto the LCD screen, which enabled us to understand why such an incorrect value was being used, quite easily and allowed us to take preventive measures to correct the issue. This methodology has also ensured that the system will behave the same in any possible situation and this behavior is correct.

3.5 POSSIBLE IMPROVEMENTS TO EXISTING DESIGN

- We have used a large number of variables, each taking up 1 or 2 bytes of the RAM memory, even if sometimes the variable is only used to hold two different values. Many of the values which are stored separately in our system, could have been implemented by using multiple bits of fewer variables, using bit manipulation. This would have tremendously reduced space costs of our system design.
- Currently, our design uses a large number of LCALL between different asm files; this has tremendously reduced performance in our system design. If all the routines could have been placed in a single asm file and jump instructions used rather than call instructions, a lot of the CPU clock cycles could have been saved. However, implementing the system in such a way would drastically reduce readability.

4. CONCLUSIONS

We were able to successfully implement a sound-based stopwatch along with all different modes of operation. The implemented design for the stopwatch also runs with a linear time complexity and constant space complexity. The number of clock cycles, program memory and most importantly, the data memory still have scope for improvement in our design. Despite this, the system designed by us meets the correctness and functional requirements.