Low-Power Keyword Spotting deployment using Voice Detection on Sensor-Tile Platform*

Abstract—**Keyword spotting (KWS) is a critical component in low power smart devices with speech-based user interactions. It is typically used as a front-end to wake up the full-scale speech recognition. Since KWS still needs to be always on, i.e. continuously listening, it has strong implications on power consumption. In this paper, voice activity detection task is activated using the build-in DFSDM analog watchdog to minimize power consumption of KWS on SensorTile. The idea is putting some parts of KWS in sleep mode when there is nothing of interest to hear, or it is just noisy voice. The part awakes the KWS as soon as voice is detected, while itself is always-on. It leads to a reduction in the KWS power consumption. The impact of the voice detection activation on power consumption is investigated through three scenarios of normal, and low power deployment with DMA streaming from beginning, or after the detection. The results show that ...**

Index Terms—**Keyword spotting (KWS), Low-Power, Voice Detection, SensorTile (SnrTl), Digital filter for sigma-delta modulators (DFSDM), Analog watchdog (AWD), Neural network (NN), Direct memory access (DMA), Microcontrollers (MCUs), STM32.**

I. Introduction

With the rapid development of mobile devices, speech-related technologies are becoming increasingly popular. In such technologies, Keyword spotting (KWS) is a critical component for enabling speech-based user interactions on low-power smart devices. It is typically used as an always-on front-end for automatic speech recognition and spoken dialog systems. Detected keywords can be used to wake up parts of system and activate the full-scale speech recognition. Therefore, KWS is required to be very energy-efficient and to be able to minimize silence or un-related background noise to limit false positives. At the state of the art, sequence of keywords can be also used as voice commands to a smart device such as a voice-enabled light [1]–[3].

Recently, neural network (NN) based methods have shown tremendous success on speech recognition tasks. They are well-suited to capture the complex, non-linear patterns from the acoustic properties of speech. This success has come after advances in the field of deep learning [2]. The networks in deep learning are quite large, requiring up to a few million multiplications every few milliseconds, as well as large memory banks for storing these weights. Since Low power practical applications are often constrained in the amount of available hardware resources, for such applications, microcontrollers (MCU), feature low-cost energy-efficient processors, are a very straightforward choice. Nevertheless, deployment of neural network based KWS on MCUs comes with the challenges of limited memory footprint and compute resources. Therefore, a resource-constrained neural network architecture exploration should be performed to find lean neural network structures suitable for the KWS deployment on MCUs [1].

ALOHA: Identify applicable funding agency here. If none, delete this.

Due to the always-on nature of the KWS, the need for minimizing its power consumption is mandatory. Some works at the state of the art attempted to reduce KWS related power by providing a hardware to trigger the audio processing part, when there is a voice detected. In these works, some parts of KWS system stays into sleep mode, when there is nothing of “interest” to hear. In those cases the voice detection, or voice activity detection task, is always-on being responsible of waking-up the other parts of KWS system as soon as a meaningful sound is heard [4] [5]. In this respect, Sensory presents a hardware for Voice Trigger Detection, embedded on STM32, called TrulyHandsfree™ Trigger [6]. Dolphin Integration also provides two types of voice trigger IP as hard IP, called WhisperTrigger™ [7]. In this work, we are using a built-in AWD in DFSDM for the purpose. What are the limitations of those works and how your approach will be different? My comment at the end of this intro is still did not met from this point of view

In this work, in order to reduce power consumption of KWS, we try to address the always-on requirement of KWS. We start with the existing deployment of KWS on tiny STEVAL-STLCS01V1 SensorTile (SnrTl) core system board with STM32L476 MCU 32-bit ARM Cortex-M4 [8].The MCUs offer the DFSDM peripheral to implement a CPU-free configurable digital filtering. The DFSDM has a built-in analog watchdog (AWD) to monitor sound level to remain within the selected high and low threshold values [9] [10]. Hence, we activate a voice detection function using the build-in DFSDM analog watchdog in SnrTl. The AWD offers a programmable threshold level to monitor voice activity. Its role is to wake-up the KWS system as soon as voice is detected, while itself runs in the always-on domain. So, the CPU is just woken up if an analog threshold is reached. Since the other blocks stay in sleep mode during the AWD “always listening mode”, the KWS power consumption is reduced. For investigating the impact of the voice detection activation on power consumption, we offer three scenarios for the KWS deployments as follows. In all configurations, a new API is presented to adjust clock tree to frequency of 12, statically.

- Normal deployment: In this deployment, the CPU in sleep mode is woken up regularly even if there is no voice for processing.

- Low power deployment with DMA streaming from beginning: In the deployment, if the sound threshold is reached, the AWD interrupt wakes up CPU in sleep mode. Since DMA streaming started from beginning, the captured data is ready in RAM, and the CPU instantly starts audio processing. In this configuration, the DMA API is modified to active DMA transfer without interrupt.
- Low power deployment with DMA streaming after AWD interrupt: In the deployment is like previous deployment. The difference is that the DMA is just clocked and activated after happening the AWD interrupt. Therefore, audio is processed with a bit delay between triggering the DMA interrupt, and starting the data gathering and saving.

In this work, we assess the power consumption, and accuracy of the offered deployments in one spectrum from very noisy to very quiet environments with respect to speech. We also explore the impact of the speech thresholds, and input size on power consumption and accuracy.

The remainder of this paper is organized as follows. Section II details the SensorTile architecture. Section III explains our contributions for the low power KWS on SensorTile using Voice detection. Section IV states our simulation environment setup, and reports the experimental results. Finally, Section V concludes this paper.

II. SensorTile (SnrTl) architecture

The tiny STEVAL-STLCS01V1 SensorTile core system board (13.5 x 13.5 mm), SnrTl, of ST Microelectronics has internal sensors, a barometric pressure sensor and a digital MEMS top-port microphone, as shown in Figure 1. SnrTl embeds an ARM Cortex-M4 32-bit STM32L476 microcontroller (MCU). The on-board 80-MHz MCU features several peripherals tuned for low-power modes, and a dedicated hardware microphone interface and ultra-low-power support [9]. In this section, the three most relevant blocks of SnrTl for our implementation are detailed, namely: the digital MEMS microphone (see Section II-A), the digital filter for sigma-delta modulators (see Section II-B), and the Direct Memory Access (see Section II-C).

A. Digital MEMS microphone

The MP34DT05-A audio sensor integrated in the SnrTl shown in Figure 1 is an ultra-compact, low power, omnidirectional, and digital MEMS microphone with a capacitive sensing element ($MEMS\ Transducer$), and an IC interface ($PDM\ Modulator$). The sensing element enables acoustic wave detection that is converted into a digital signal, in a pulse density modulation (PDM) format, by the IC interface. The PDM format is equivalent to a sigma-delta modulated signal, which is supported by the DFSDM.

The digital microphones require a microphone clock input signal (CLK), and DATA output line ($DOUT$). The clock signal defines the microphone output data rate into the DFSDM, which can be provided by the DFSDM output clock signal. The DATA output line sends the raw converted digital data, which is processed by the DFSDM [9].

B. Digital filter for sigma-delta modulators (DFSDM)

The DFSDM is one of the peripherals of the STM32L476 MCU within the SnrTl implementing CPU-free configurable digital filtering. It averages a fast rate input serial stream, and produces a parallel, lower rate, data output with higher resolution. The DFSDM filter ($Digital filter(Sinc)$) features a set of configurable parameters, e.g. filter order, and oversampling ratio, that allows to tune the output resolution and data rate, in order to meet the application requirements. DFSDM functionality cannot be reduced to digital filtering only, but consists in a complete digital peripheral that handles the overall A/D conversion. The $Serial\ transceivers$ block receives serial data from the external sigma-delta modulator. It offers SPI and Manchester serial protocol formats with configurable rising/falling sampling clock edge to support most of the sigma-delta modulator types. It also supports PDM, which means it can convert the MP34DT05-A microphone output. The optional $Integrator$ works as a simple adder. It sums up a given number of samples provided by the filter output. In the figure, the $Output\ data\ unit$ performs a final correction that consists in shifting the bits to the right and applying an offset correction on the data coming from the integrator.

The DFSDM also featres additional functions such as analog $Watchdog$ (AWD). In particular, the AWD purpose is to control the excursion of the DFSDM data, when the signal exceeds predefined thresholds the AWD may trigger the microcontroller CPU (interrupt) or other peripherals (break signal).

C. Direct memory access (DMA)

The DMA of the STM32L476 MCU handles both high-speed data transfer between peripherals and memory, as well as memory to memory. Data transfer can be handled by the DMA without any CPU intervention, which can focus on other operations. As seen in in Figure 1, the DMA controller performs direct memory transfer by sharing the system bus ($Bus\ matrix$) with the Cortex®-M4 core (CPU). Therefore the DMA request may stop the CPU access to the system bus for some bus cycles, when the CPU and DMA are targeting the same destination (memory or peripheral). The $Bus\ matrix$ implements round-robin scheduling, thus ensuring at least half of the system bus bandwidth (both to RAM and $Peripheral$) for the CPU. After an event, the $Peripheral$ sends a request signal to the DMA controller. The DMA controller serves the request depending on the channel priorities. As soon as the DMA controller accesses the $Peripheral$, an acknowledge is sent to the $Peripheral$ by the DMA controller. The $Peripheral$ releases its request as soon as it gets the acknowledge from the DMA controller. Once the request is de-asserted by the peripheral, the DMA controller release the acknowledge. If there are more requests, the peripheral can initiate the next transaction [11].

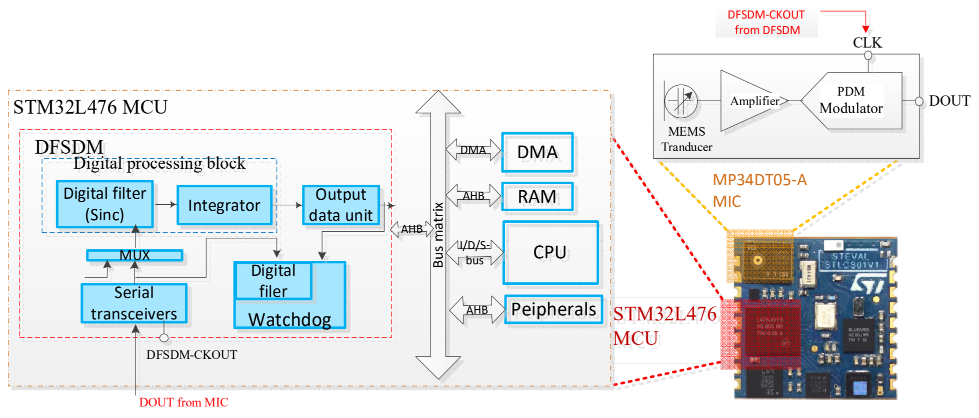

Fig. 1. General view of SensorTile architecture.

III. Low power KWS Deployment on SensorTile using Voice detection

The contribution of our work is the implementation and analysis of different system configurations to perform KWS on the SnrTl, with the final purpose of reducing power consumption in KWS deployments. Section III-A briefly presents the application, while Section III-B describes the processing chain. The core of our work is presented in Section III-C where three different system configurations are discussed. The idea behind these different configurations is that using the build-in DFSDM-AWD in the SnrTl, it is possible to save power consumption. DFSDM-AWD voice detection function executed in an always-on manner, removes the needs of having an always-on KWS parts when there is no voice to be captured. DFSDM-AWD voice detection acts as a trigger for the rest of the system, and wakes it up as soon as voice is detected. somewhere here you should present the power modes, otherwise it will not be clear afterwards.

A. KWS

A typical KWS system consists of a feature extractor and a neural network-based classifier. Feature extraction using LFBE or MFCC involves translating the time-domain speech signal into a set of frequency-domain spectral coefficients, which enables dimensionality compression of the input signal. The extracted speech feature matrix is fed into a classifier module, which generates the probabilities for the output classes [1]. In this work, the input speech signal of length L, e.g. 1000ms, is framed into overlapping frames of length l, e.g. 40ms, with a stride s, e.g. 40ms, giving a total of $T = \frac{L \times l}{s} + 1$, e.g. 25, frames. From each frame, F, e.g. 10, speech features are extracted, generating a total of $T \times F$, e.g. 250, features for the entire input speech signal of length L, saving in the buffer called $kws \rightarrow mfcc\_buffer$. In a real-world scenario where keywords need to be identified from a continuous audio stream, the process has been performed on small interval of the input speech signal with the length of a recorded window. The window length is considered double frame length which is 80ms.

So, every 80ms, DMA interrupt for filling the half parts of ping-pong buffer happens. In both DMA interrupt service routine, i.e. half and complete transfer, audio process is called. In the audio process, the recorded audio with the format of PCM is transferred to the space of program. Then a function is called to extract the features for the two recorded frames, and fill the two columns of the feature matrix. Another function is called to classify the input speech signal including the 23 previous frames with the two new added ones all together using Deep Neural Network (DNN). Finally, a posterior handling module averages the output probabilities of each output class over a period of time, improving the overall confidence of the prediction. Finally, the probability of the detected command is assessed to report if is more than a specified threshold.

B. Digital audio data flow: acquisition and processing

This subsection explains the generic processing chain for digital audio acquisition and processing shown in Figure 2. The audio signal is captured by the microphone, in our case the digital MEMS microphone explained in Section II-A. The microphone outputs a 1-bit digital PDM signal ($PDMData$) with frequency that can vary between 1 to 3.25 MHz. The microphone output data rate is determined by the DFSDM clock output signal shown with the dashed arrow comes from DFSDM to microphone in the figure ($Clock\ Signal$). Then the data is acquired through the DFSDM serial transceiver, connected to the external sigma-delta modulator of the digital microphone.

The DFSDM converts the data into the PCM raw ($PCMData$). The digital filters in the DFSDM perform CPU-free filtering to average the 1-bit input data stream from the SD modulator into higher output resolution at lower sample rates [10].

After the conversion, data are transferred by the DMA to a system RAM buffer to reduce the software overhead.

The PCM raw data are processed by KWS applications through $MFCC$, DNN, and $WFST/Viterb$.

The detected command along with any other meaningful processed information (in the specific case of our KWS application the probability percentage) are sent over serial wireline

USB connection to the monitor on the $PuTTy$ application as shown in the figure. The general flow depicted in Figure 2 can be considered as the baseline flow. DMA transfers and KWS are always-on and no DFSDM-AWD voice detection function is executed.

C. Low-power KWS

In this subsection, we detail the activation of voice detection functions in SnrTl that leverages on the embedded AWD in the DFSDM. Two variations of the baseline processing flow have been set-up. Indeed, on the one-hand, voice detection can be used as a trigger just for the computation, considering the DMA transfer in the always-on domain; on the other, DFSDM-AWD voice detection can be used as a trigger for the DMA clock activation and transfer too. DMA clock gating and saving DMA transfers has an impact on power consumption and threshold selection. In fact, if DMA is always-on, transfers from the DFSDM to RAM continuously take place, meaning that the data in memory are always ready to be processed as the the voice detection trigger happens. No sample loss happens, out of bounds data are transferred too due to the fact that the AWD triggers the processing only. Such a complete samples availability in memory implies power waste due to out of bound transfers. On the contrary when DFSDM-AWD voice detection triggers DMA transfers, rather than computation only, out of bound data are not transferred to memory, saving a potentially considerable amount of power. Point is that in this second case, as it will be clarified by the example below, there will be a certain amount of sample loss, which may imply the need for lower AWD thresholds to guarantee correctness in the KWS. Below the set-up of the different configurations, namely baseline (B), KWS triggered (KWS-T) and DMA plus KWS triggered (DKWS-T), is detailed.These three scenarios for KWS implementation on SnrTl are meant to assess the impact of the voice detection activation on power consumption, and are summarized in Figure 3.

As already said, the activation of voice detection functions is realized using an embedded AWD in the STM32 DFSDM. The AWD has two programmable thresholds (low and high), which are defining a normal operation window. Outside of this window, the AWD interrupt, and its callback appears [11]. The programmable threshold must be chosen, taking into consideration the product application and its environment. If this latter is noisy a large watchdog window is preferable. In quite, or almost quite, environments a reduced window size can be set.

The three configurations in Figure 3 correspond to different usage of the DFSDM, AWD, and DMA. In the figure, in the first column on the left, you can see an audio pattern, in the example the waveform represents the "yes" command followed by "silence". Simultaneously Figure 4 is showing MCU modes, active blocks, and events corresponding to three configurations. In the figure, M1 and M3 indicate sleep mode with DMA transferring, and DMA clock gating respectively, as well as M2 shows run mode in the frequency of 12Mhz with the voltage of range2. The events include in E0 for start system, E1 for DMA interrupt for filling buffer, EOP for ending KWS process, and E2 for AWD interrupt. The frequency has been adjusted in somehow EOP happens before E1 in all configurations. In the first scenario shown in the second column of Figure 3 and labelled by (1), and first row of Figure 4, corresponds to the baseline processing flow of Figure 2. The core stays in sleep mode, i.e. M1, the DFSDM collects the captured data regularly, and the DMA stores the data in the system RAM. When the DMA interrupt for filling the buffer, i.e. E1, occurs, the CPU performs KWS analysis on the collected data in mode M2. In this deployment, the CPU is woken up regularly by the DMA interrupt even if there is no real voice to be processed. to make it more technically sound rather than informal discussion you should put here the relationship between mode defined from you and Sensortile modes.

In the two KWS-T and in the DKWS-T scenarios, the DFSDM still runs in regular conversion mode, but the AWD is enabled to detect input signal level. When thresholds are reached, the AWD interrupt occurs, i.e. E2 . In Figure 3, the red dash line is showing the *High Threshold* that is reached while the command "yes" is received. In the DKWS-T scenario, shown in the third column of the figure and labelled by (2),and the second row of Figure 4, the DMA clock activation and transfer is started after the voice detection event , i.e. E2, while before E2, MCU is in M3 mode. When AWD interrupt occurs, the converted data are gathered, and saved in a RAM buffer by DMA. This procedure requires to disable the regular DFSDM conversions, and DFSDM conversions with DMA transfer and DMA interrupt are enabled. Therefore, a certain delay has to be taken into account between AWD interrupt and the beginning of DMA data transferring. Such delay corresponds to the sample loss. The DMA interrupt, i.e. E1, which corresponds to *Start Processing*, is generated when the buffer gets full. In fact the transfer-complete interrupt is occurred, when the half or full bounds of the ping-pong buffer are hit. After hitting the buffer bounds, the interrupt callback in M2 mode calls the audio process to detect the commands based on neural network.

To avoid sample loss, the KWS-T scenario has been defined. In the KWS-T scenario, shown in the fourth column of the figure and labelled by (3), and third row of Figure 4, the DMA transfer is always active, and does not depend on AWD interrupt. In this scenario, CPU is in M1 mode and DFSDM regularly converts the data, and DMA regularly fills the buffer, without generating any interrupt after hitting the buffer bounds. So there is need an API modification for dis-activating the event of E1 during M1 mode. Data are available in memory before the AWD interrupt, without any delay. Once the AWD interrupt occurs, i.e. E2, it is checked which half of the buffer has been completed. By knowing that, the audio process is done on that half in M2 mode, and starts the DMA transfer this time with the event of E1. In both scenarios, audio processing is almost stopped after detecting the current commands.

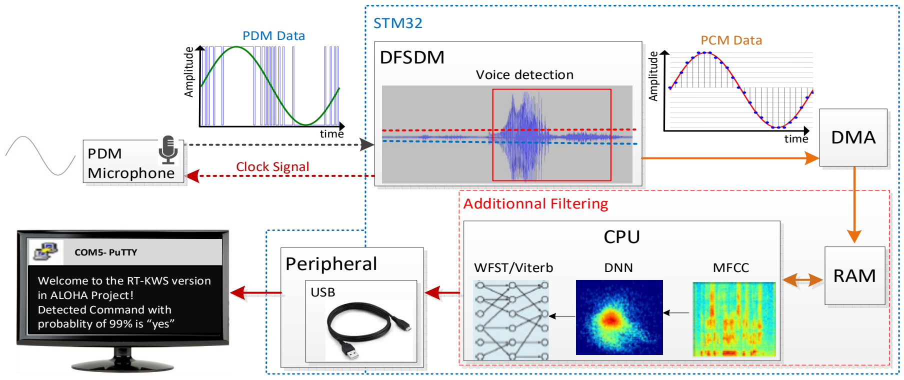

Fig. 2. Baseline System Configuration - acquisition and processing.

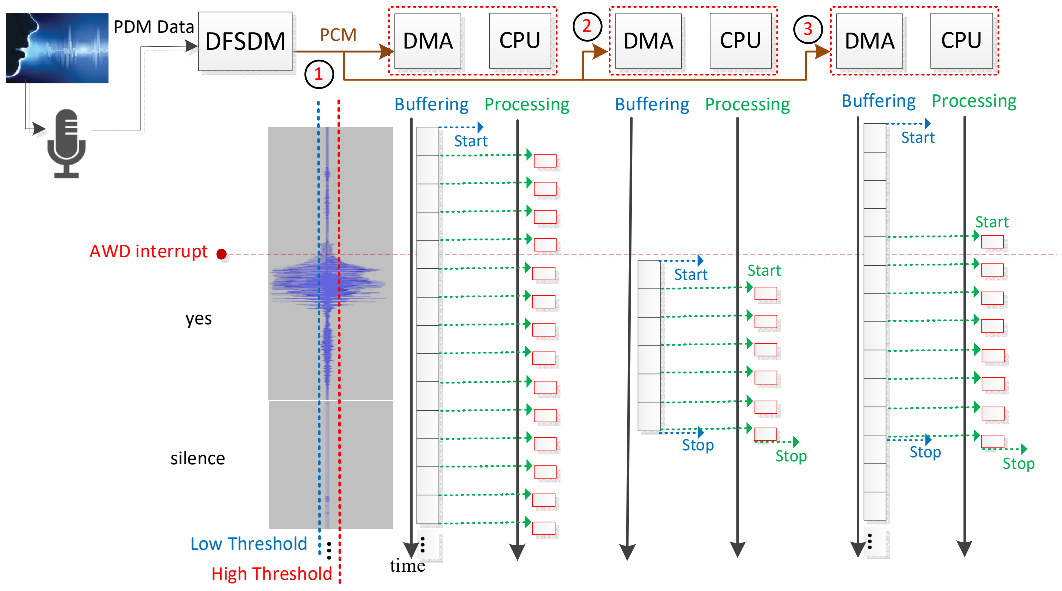

Fig. 3. Conceptual illustration of low power KWS with voice detection activation through configuring DFSDM, AWD, and DMA.

IV. Simulations setup, results, and analysis

explaining Simulations setup (how to make inputs and ....)

A. Threshold definition

AWD thresholds have to be defined in a way that keyword recognition is still guaranteed. In the following, we present an empirical approach, depicted in Figure 5, for thresholds definition. Since there are 105,829 utterances from 2,618 speakers for each command in the reference database [12], one wave for each command has been randomly chosen, and inserted in CMD_SET. Please note that in the database, commands have some background noise and unrelated speech. Moreover the commands are being played in a real environment. Therefore, the threshold definition process takes into account a realistic situation.

As seen in the figure, thresholds per command (CMD_THR) in the command set (CMD_SET) are initialized to $\pm 5k$. Then each command ($cmd[i]$) is played separately close to the microphone in a real environment. If the command is recognized correctly, the thresholds are incremented ($\pm 5k$ per iteration). When the KWS is not capable of recognizing the given command anymore, the values of the thresholds at the previous iteration ($THR \pm 5k$) are stored as the reference thresholds for the current command ($cmd[i]$) in the position i of the vector of thresholds per command (CMD_THR). The definition process is re-started for next command, until no commands are available in the CMD_SET.

The results including the maximum possible thresholds for each command have been reported in Table I. Please note that each value reported in the table is symmetric for positive and negative AWD thresholds. As seen in the table, the maximum and minimum thresholds, i.e. $\pm 45k$, and $\pm 25k$, belong to "go", and "no", respectively.

B. Simulations setup, results, and analysis

At the first experiment, KWS audio process time, when the the length of a recorded window is $80ms$, is investigated to

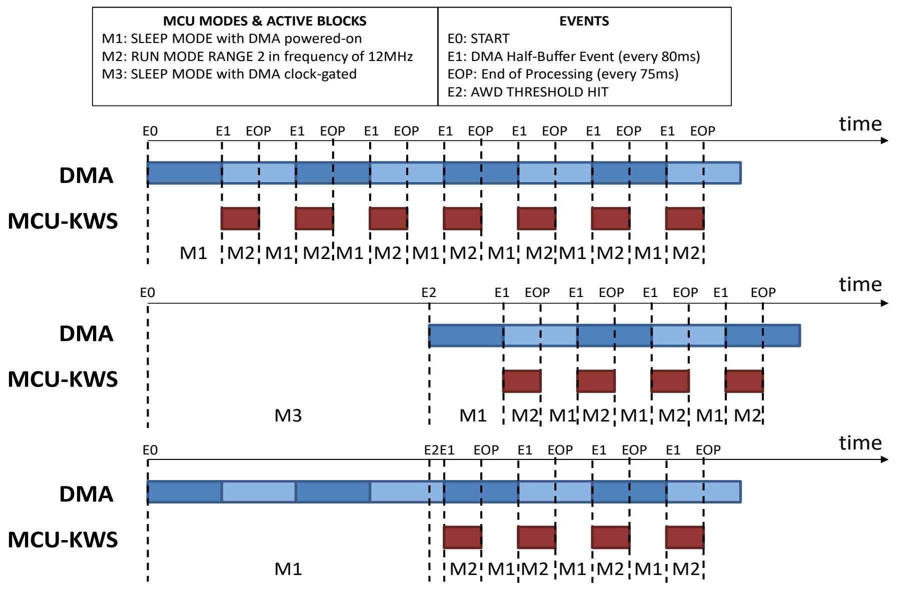
MCU MODES & ACTIVE BLOCKS
M1: SLEEP MODE with DMA powered-on
M2: RUN MODE RANGE 2 in frequency of 12MHz
M3: SLEEP MODE with DMA clock-gated
EVENTS
E0: START
E1: DMA Half-Buffer Event (every 80ms)
EOP: End of Processing (every 75ms)
E2: AWD THRESHOLD HIT
time
DMA
MCU-KWS
M1
M2
M3
E0
E1
EOP
E2

Fig. 4. MCU modes, active blocks, and events in three configurations.

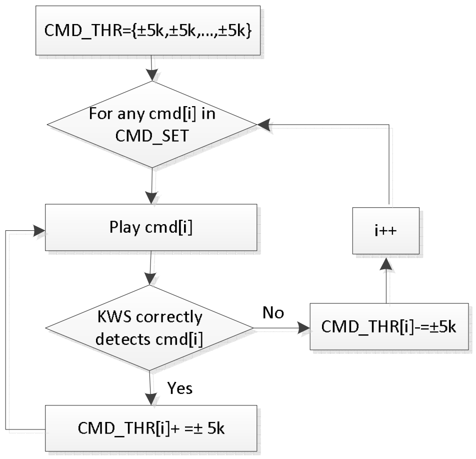
CMD_THR={±5k,±5k,...,±5k}
For any cmd[i] in CMD_SET
Play cmd[i]
KWS correctly detects cmd[i]
No
Yes
CMD_THR[i]-=±5k
i++
CMD_THR[i]+ =± 5k

Fig. 5. Threshold definition process.

TABLE I
MAXIMUM THRESHOLD (MXT) (±K) FOR EACH COMMAND (CMD)

Cmd	*yes*	*no*	*on*	*off*	*lft*	*rght*	*up*	*dwn*	*stp*	*go*
MxT	40	25	40	35	25	30	40	35	35	45

optimize the frequency at the aim of saving power. The process time is tabulated in Table III in two frequency of 80Mhz and 12Mhz. As seen in the table, The time improvement in compilation optimization of O3, and Os for size are considerable. So The process time with the optimization of O3 has been considered as a reference in the whole paper. since the recorded window is 80ms, the process time in frequency of 12Mhz does not violate the required time for filling DMA buffer in compile optimization of $O3$. In this part, there is a need to create new API to finely set the frequency of the system to 12Mhz.

TABLE II
KWS AUDIO PROCESS TIME WITH RESPECT TO FREQUENCY AND COMPILE OPTIMIZATION

Frequency	*80Mhz*	*12Mhz*
Without Opt.	42.56	28.37
With Opt.(O3)	11.12	74.2

TABLE III
KWS AUDIO PROCESS TIME WITH RESPECT TO FREQUENCY AND COMPILE OPTIMIZATION.

Frequency	Optimization		
	no Opt.	*Os*	*O3*
80 Mhz	42.56	11.55	11.12
12 Mhz	28.37	77	74.2

o measure power, WaveRunner 6 Zi and 12-Bit Oscilloscopes is used. In the Oscilloscope, the maximum time interval for capturing voltage is 3.2 second. To investigate power in the proposed configurations in sleep mode, we set the threshold in DKWS-T and KWS-T to the maximum value, i.e. 8388607, and measure power in the frequency of 12Mhz for the configurations. The results have been tabulated in Table IV. As shown the second configuration, i.e. DKWS-T, with clock gating has the lowest power in sleep mode.

TABLE IV
POWER CONSUMPTION OF CONFIGURATIONS IN MILIWATT IN SLEEP MODE IN THE FREQUENCY OF 12MHZ

Configurations	*DKWS-T*	*KWS-T*
Power(mW) in sleep mode	22.69	28.63

To investigate the power of the proposed configurations (B, KWS-T and DKWS-T) in active mode, we set the threshold in DKWS-T and KWS-T to the minimum value, i.e. 25k, obtained in the previous subsection, and measure power in the frequency of 12Mhz for the configurations. To perform such an experiment and measuring power, a wave composed of commands has been played in front of microphone. Then the voltage has been captured for 2 second to get average. The results have been tabulated in Table V. As shown in the table, power in active mode in all configurations are close.

TABLE V
POWER CONSUMPTION OF CONFIGURATIONS IN MILIWATT IN ACTIVE MODE IN THE FREQUENCY OF 12MHZ

Configurations	***B***	***DKWS-T***	***KWS-T***
Power(mW) in active mode	30.948	29.821	29.945

C. Results, and analysis

The class file is designed for, but not limited to, six authors.

TABLE VI
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

[1] L. Lai V. Chandra Y. Zhang, N. Suda. "hello edge: Keyword spotting on microcontrollers". *arXiv e-prints*, Nov. 2017.

[2] Zhong Qiu Lin, Audrey G Chung, and Alexander Wong. Edgespeechnets: Highly efficient deep neural networks for speech recognition on the edge. *arXiv preprint arXiv:1810.08559*, 2018.

[3] Sanjay Krishna Gouda, Salil Kanetkar, David Harrison, and Manfred K Warmuth. Speech recognition: Keyword spotting through image recognition. *arXiv preprint arXiv:1803.03759*, 2018.

[4] M. Price, J. Glass, and A. P. Chandrakasan. A low-power speech recognizer and voice activity detector using deep neural networks. *IEEE Journal of Solid-State Circuits*, 53(1):66–75, Jan. 2018.

[5] M. Shah, J. Wang, D. Blaauw, D. Sylvester, H. Kim, and C. Chakrabarti. A fixed-point neural network for keyword detection on resource constrained hardware. In *IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, Oct. 2015.

[6] L. Luca Spelgatti. Low power microphone acquisition and processing for always-on applications based on micro-controllers. Sensors Expo and Conference, 2017.

[7] Dolphin Integration. *Leverage always-on voice trigger IP to reach ultra-low power consumption in voice-controlled devices*, 1 2016. Whisper-Trigger™ - MIWOK™ - Technical Paper.

[8] N. Suda. Keyword spotting for microcontrollers. https://github.com/ARM-software/ML-KWS-for-MCU, 2018.

[9] STMicroelectronics. *User manual: Getting started with the STEVAL-STLKT01V1 SensorTile integrated development platform*, 3 2019. Rev. 5.

[10] STMicroelectronics. *Application note: Getting started with sigma-delta digital interface on applicable STM32 microcontrollers*, 3 2018. Rev. 1.

[11] STMicroelectronics. *Reference manual: STM32L4x6 advanced ARM®-based 32-bit MCUs*, 6 2016. Rev. 4.

[12] P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.