W562XXX DESIGN GUIDE

Winbond Electronics Corp.

2-tone Melody+ADPCM Voice Synthesizer (BandDirector™ Family)

INTRODUCTION

The W562xxx is a family of multi-engine speech synthesizers. These synthesizers incorporate part of the following parts into a single chip: a simple 4-bit uC core (including RAM, register file, timer, interrupt control logic, ALU, and stack), speech synthesizers and Dual tone melody generator.

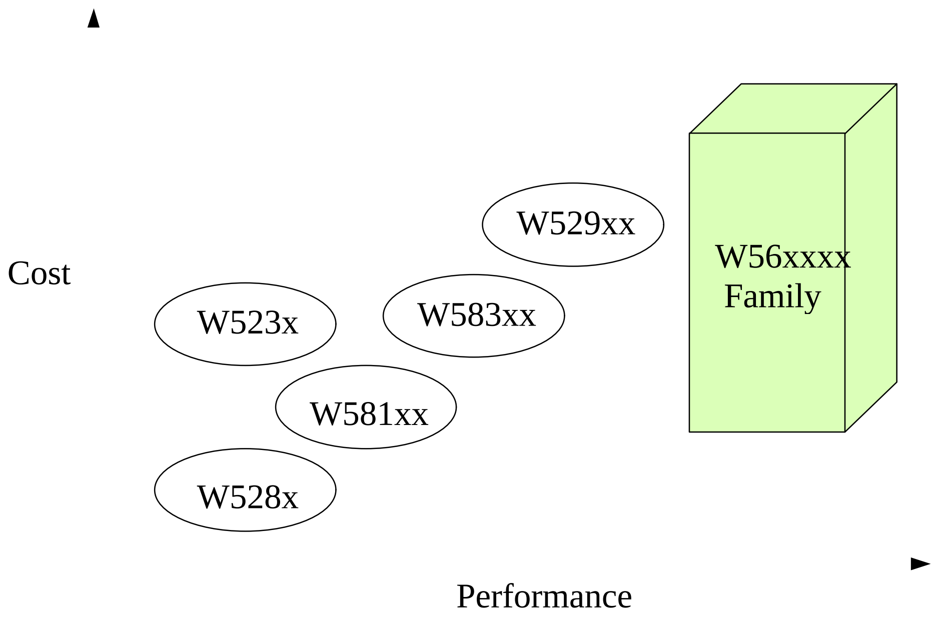


Figure 0-1. Speech Synthesizer Spectrum

As shown in Figure 0-1, the W56xxx satisfies those applications of parallel processing requirements and especially of good quality melody effects. On the other hand, customers select PowerSpeech series (W528x, W523x, W581xx, W583xx, and W529xx) for the sake of cost issues.

Also noted in Figure 0-1, the W56xxx family shows the same performance while spans into a wide range of cost structure. That's because the different components selected for a certain family member to fit specific applications.

Typical applications for the W56xxx family are listed below:

- Talking clocks
- Toys
- Direct-Mail Advertisements
- Computer-aided instruction
- Games
- Edutainment toys
- Gifts
- Warning Systems

Winbond supports the development for the W56xxx family as usual -- user friendly developing environment. Several development tools are provided in order to cut the time required to develop a code. They are ICE (In Circuit Emulator), Speech Coding System Version 7.53.2, and demo boards.

The W56000 chip is used for the In Circuit Emulation for all of the family members. All of the derivatives of this family is a sub-set of the W56000 chip.

GENERAL DESCRIPTION

The W562xxx is a sub-set of the W56000. It consists of a 4-bit uC, two ADPCM synthesizer and one dual tone Melody generator. The internal 4-bit uC is composed of a 64-nibble RAM, and an 8-bit timer.

The W562xxx is one of the derivatives of the W56000 family (the MIDI speech products with multi-engine capability). It is capable of processing µC, speech synthesis, and dual tone Melody generation in parallel. The internal 4-bit µC is a simple engine to execute instructions of up to 12 KIPS (Kilo-Instructions Per Second). Other engines are activated in parallel with the µC to implement specific tasks, like speech syntheses and dual tone Melody generation. The dual channel operations (synth1 and synth2, or synth and dual tone Melody) is implemented by dedicated H/W only with the help of synth interrupts for concatenating voice segments and melody phrases automatically. The efforts in programming the speech equation is much like that of the PowerSpeech. User-friendly programming style is thus preserved under Winbond's speech coding system.
The W562xxx is equipped with the W56000 ICE system, customers' programs can be debugged efficiently and effectively than ever.

PART NO.	**W562S08**	**W562S10**	**W562S12**	**W562S15**	**W562S20**	**W562S25**	**W562S30**
Sec.	8	10	12	15	20	25	30
ROM (Kbit)	256	288	320	480	576	672	768
PART NO.	**W562S40**	**W562S50**	**W562S60**	**W562S80**	**W562S99**	**W562M02**	-
Sec.	40	50	60	80	100	120	-
ROM (Kbit)	1216	1376	1536	2304	2688	3072	-

There are 2 bodies in W562M-xx family for two chip solution, list as following.

Product #	Duration	ROM	Chip Set Configuration
W562M-04	4 minutes	6 Mbits	W56000 + W55M06
W562M-05	5 minutes	8 Mbits	W56000 + W55M08

Possible applications are:
- Programmed voice synthesis with background music or speech.
- I/O interactive voice synthesis to accompany with background music or speech.
- Q&A games.
- Edutainment toys.

FEATURE

- Multi-engine processor parallel management with µC, speech and Dual Tone Melody.
  – µC // (Synthesizer1 or Dual Tone Melody) // Synthesizer2 (//: in parallel)

– µC, with basic ALU, 64-nibble RAM (including 8 working registers) and an 8-bit timer.
– Synthesizer1 capable of voice syntheses with Sample rate @ 4.8/6/8/12 KHz
– Synthesizer2, same as synthesizer1.
– Dual-tone Melody D/A output with 3 level volume control

- Wide operating voltage range: 2.4 to 5.5 volts
- Low power consumption (V_{DD} = 5 Volt)
  – Standby current < 1 µA
  – Operating current < 1 mA
- Main oscillator: 3 MHz, Ring oscillation
- Input/ Output port
  – Port for input only: 1 port/ 4 pins
  – Input/ Output ports: 2 ports/ 8 pins
  – Port for output only: 1 port/ 4 pins
  – Can offer a direct row and column matrix of up to 72 (8×9) keys
- Interrupts
  – Internal interrupts: Timer
  – External interrupts: TG (port 0, port1), POI (Power On Initialization)
  – Priority: POI > TG > Timer
- Melody + Voice outputs for DAC
- TG interrupt provided
  – Share TG interrupt for Port0/Port1 input
  – Global TG interrupt enable controlled (bit3 of the IER register)
  – Individual interrupt enable controlled (PER0 and PER1 registers)
- Built-in 8 bit programmable down count timer
  – One of two internal clock frequencies can be selected
  – Desired Timer interval = (preset value+1) * 1/F_T
  (F_T: 32 Hz or 32 KHz dependent on the bit0 of the MODE register, at Fosc = 3 MHz)
- A total of around 64,000 instructions can be used in the program
- Powerful instruction set:
  – Arithmetic: ADD, ADDC, SUB, SUBC, INC, DEC, SETB, CLRB
  – Logic Operation: AND, OR, XOR, NOT
  – Shift & Rotate: RORC, ROLC, SHRC, SHLC
  – Date move: LD, LDR, MV
  – Branch: JP, JB0, JB1, JB2, JB3, JZ, JNZ, JC, JNC, JBZ1, JBZ2, CJNE, CJE, DJNZ, DJZ
  – Subroutine: CALL, RTN, RTI
  – Others: NOP, END, EN INT, DIS INT, PLAY CH1, PLAY CH2, STOP CH1, STOP CH2
- 8-level STACK shared by CALL, Timer, Synthesizer and TG
- Multi-tasking operation via interrupt for automatic voice segment concatenation

- – Melody or Speech voice can be easily concatenated with symbol "+"
- – Example: PLAY CH1, H4 + Melody1 + Speech1 + Speech2 + Melody2 + T4

  The DAC of the W562xxx will play Melody1, Speech1, Speech2 and Melody2 sequentially

- The length of the voice segment is unlimited
- Speech section control
  - – Sample rate control (4.8K/6K/8K/12K)
  - – Example: PLAY CH2, H4 + speech1_S + T4; S: define the sample rate
- Melody section control
  - – Background music D/A output bits selectable for volume control (6/7/8)
  - – Example: PLAY CH1, H4 + Melody_xxB + T4; B: define the melody output bits
- Dual tone melody with
  - – XM3: Triple harmonic effect
  - – 3 kinds of percussion effects
  - – 6 beats
  - – 41 pitches from G3# to C7
  - – 16 kinds of tempo
  - – The number of the score and note are unlimited
- Provide IR 38 KHz carrier
  - – TXF.0 = 0/1 disable/enable IR carrier
  - – TXF.1 = 0/1 output carrier with P2.3 low/high active
- Provides ICE (In Circuit Emulation) system for easy debugging
  - – Free Run
  - – Stop Run
  - – Program Reset
  - – Step Into
  - – Step Over
  - – Go To Cursor
  - – Break point
  - – Register read/ modify

W562xxx PAD DESCRIPTION

Name	I/O	Description
P0.0 - P0.3	I	TG pins, internally pulled high.
P1.0 - P1.3	I/O	I/O multiplexed port 0. Interruptable port if selected as input.
P2.0 - P2.3	I/O	I/O multiplexed port 1. status port only if selected as input.
P3.0 - P3.3	O	Output pins.
DAC	O	Current output of mixing channel 1 and channel 2 for driving an external speaker.
TEST	I	Test pin, internally pulled low.
OSC	I	Connect ROSC to VDD to generate the 3 MHz master frequency
/RESET	I	Reset all, functions as POR, internally pulled high.
VSS	-	Negative power supply.
VDD	-	Positive power supply.

BLOCK DIAGRAM

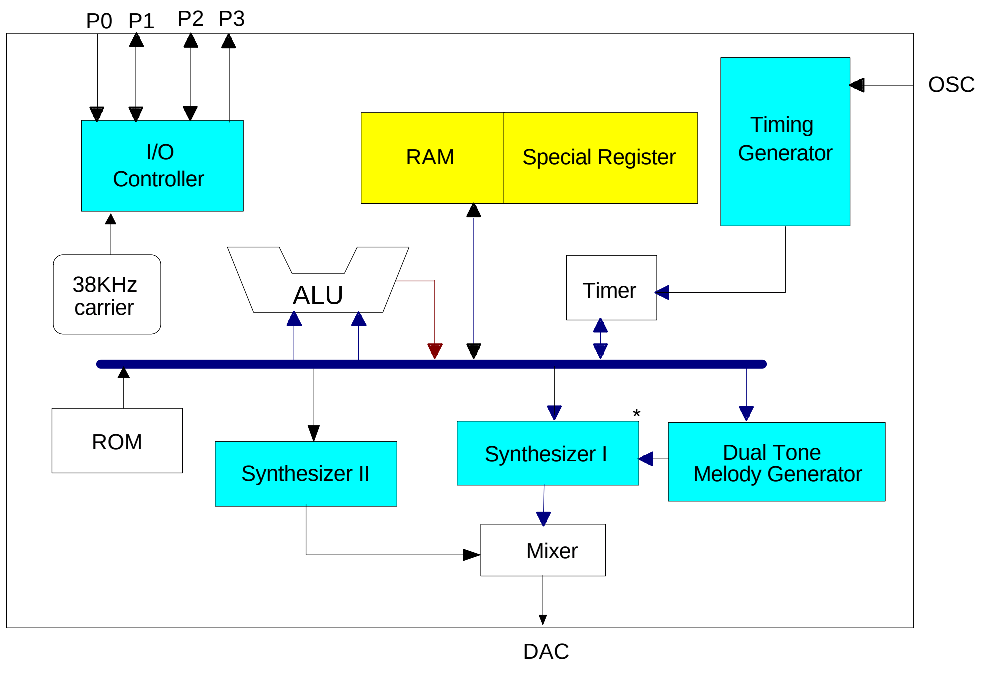


Figure 0-2. Block Diagram of W562xxx

FUNCTION DESCRIPTION

The W562xxx is basically a subset of the W56000 ICE chip, which is a µC-based speech processor with multi-tasking capability to implement the program control, voice synthesis, and dual tone Melody generation in parallel.

There are maximum 8 external TG interrupts (P0 & P1) which are serviced upon the encountering of the falling-edge triggers. These TG input are different from Winbond's previous PowerSpeech owing to the interrupt essence, which means these TG inputs don't overwrite the current operation, but interrupt instead. After completion of the common ISR, the unfinished program that is interrupted by TG will be continued. The POI is another interrupt that executes automatically upon power up or depression of the RESET pin. A total of 8 x 9 keypad matrix can be formed by configuring 8 inputs (P0 & P1) , 8 outputs (P2 & P3) and ground without external components.

The W562xxx can synthesize dual-channel voices with different sample rate or voice synthesis plus melody generation independently, meanwhile the control program proceeds to execute regardless of the dual-channel operation. For synchronization among the control program and voice syntheses, two busy flags (BZ1 & BZ2) concerning the status of the dual channels are available. Also, at the end of each voice segment synthesis or melody phrase generation, the H/W channel interrupt is generated to finish the playback of the channel list. After the completion of a channel list, each channel must be terminated by an identifier to mark "End Of List (EOL)." Should the EOL mark be encountered, the specific channel is said to enter the IDLE state, where the busy flag turns down at that moment.

The sample rate used for each channel may be different to make efficient use of the memory space and to have an acceptable voice quality. For each channel list, there could be delays, which could be made by inserting silence of certain length, between successive voice segments or melody phrases to allow for better synchronization of the dual channel outputs. Because of the parallel processing capability of the W56000 family, customers now have the ability to change status of the output pins during the period of channel operation without sacrificing the continuation of voice combinations.

The ALU together with the 64-nibble RAM of the W56XXX offers customers a great deal of flexibility to achieve various kind of program controls for different applications.

The W562xxx program can be developed by customers themselves quite easily as **PowerSpeech**™ products. The developing environment is much the same with current Winbond's speech coding system. Besides, another powerful debugging tools, the W56000 ICE system, is provided to assist program development. Less effort, but high output is always the programming guideline for the W56000 family.

P0.0 - P0.3

The P0 port is interruptable trigger inputs with separate internal pull-up devices. This port is usually regarded as TG1 - TG4 in previous PowerSpeech products. No internal debounce circuits is available with P0, since the bounce is to be eliminated by means of user's program. A common ISR, for P0 & P1 (provided that P1 is selected as input port), will be invoked once a low-going edge is sensed on the port, the program is then used to further differentiate which pin of the port is actually pressed down.

P1.0 - P1.3

These pins can be selected to be either inputs or outputs, depending on the status of IOR1 (I/O register 1). By setting the PCR1 (Pin Configuration Register 1), each selection can also be defined as passive pull-up or floating for input pins, or defined as inverter or open drain outputs. See Control Registers for further information. The P1 port is interruptable and invokes the same ISR as P0, if selected as input.

P2.0 - P2.3

These pins can be selected to be either inputs or outputs, depending on the status of IOR2 (I/O register 2). By setting the PCR2 (Pin Configuration Register 2), each selection can also be defined as passive pull-up or floating for input pins, or defined as inverter or open drain outputs. See Control Registers for further information. The P2 port is a status port (not interruptable, users have to move it to internal RAM for further processing), if selected as input.

P3.0 - P3.3

The P3 is an **inverter type** output port.

VDD & VSS

VDD is the positive power supply, while VSS is the negative power supply. In order to prevent possible power noises generated during motor driving from interfering the proper operations of the internal POR circuit, which is essential to the POI(Power On Initialization) process of the PowerSpeech synthesizers, two approaches are being adopted in the W562xxx one is to use the /RESET pin, which is described in detail elsewhere, to fully reset the internal circuit for a brand new start; the other is to place VDD and POR circuit apart as far as possible to reduce the possibility of noise induction.

In order for the W562xxx to process the POI correctly, the VDD has to drop low enough and rise quite quickly to initiate the internal POR circuit for generating the required pulse. Special care goes to the discharge of VDD to nearly ground level to ensure proper operations.

TEST

The TEST pin is used solely for test purposes. It is internally pulled low by an NMOS device with a 200KΩ resistance to prevent floating-gate conditions.

/RESET

Active low reset input with an internal pull-high resistance of 500 KΩ. The falling edge of the /RESET pin will reset the W56XXX totally, just like the POR condition. Right at the rising edge of the /RESET input, the W56XXX starts to awake and proceeds to undergo the POI process.

Originally, the /RESET pin is used to function as a last resort to rescue the POR failure issue, which is encountered in some occasions where the internal POR circuit of the W56XXX can't operate properly. If customers failed to discharge the VDD to ground level and re-power up the W56XXX, it may function abnormally, causing unpredictable operations. Users may then reset the W56XXX by sending a pulse through the /RESET pin to re-start the operation from the very beginning: POI. Maybe, this is the safest way to get around all the annoying POR issues.

DAC

The DAC pin is a current-type voice output, which is connected to the output of the internal D/A converter. The full scale output of the 8-bit D/A converter is 5 mA, which is able to drive the external 8-W speaker through the amplification of a low-power NPN transistor with a β of around 120 - 160. (Usually, the selection of an 8050D transistor is appropriate.)

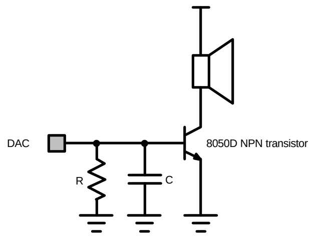


Figure 0-3. DAC Current-type Voice Output

The shunt resistor R in Figure 0-3 is used to reduce the current that enters into the base of the NPN transistor for driving the external speaker without distortions, which may occur in the above simple scheme.

Distortions result from two facts: one is the saturation phenomenon of the transistor due to large I_B, the other is the introduction of R that cuts small signals too much out of the original waveform. A typical value of the shunt resistor is around 470Ohm - 1KOhm . The smaller the resistance, the smaller the current enters the transistor and vice versa. Users have to trade off what value of R could be added without causing these two kind of distortions.

The capacitor C is used for low-pass filtering the unwanted high-frequency noises that are generated from D/A converters during sample transitions. Users may adjust the capacitance to reach a better perceptual hearing. It could be simply omitted without affecting the voice quality too much.

OSC

The master frequency (3 Mhz) of the W562xxx can be generated by ring oscillator. Connect OSC to VDD via ROSC.

ABSOLUTE MAXIMUM RATINGS

ITEM	SYMBOL	CONDITIONS	RATED VALUE	UNIT
Power Supply	VDD - VSS	-	-0.3 - +7.0	V
Input Voltage	VIN	All Inputs	VSS-0.3 - VDD+0.3	V
Storage Temp.	TSTG	-	-55 - +150	°C
Operating Temp.	TOPR	-	0 - +70	°C

NOTE: Operating the device under conditions beyond those indicated above may cause permanent damage or affect device reliability.

ELECTRICAL CHARACTERISTICS

DC PARAMETERS

(V_{DD}–V_{SS} = 3.0V, Fm = 3 MHz, T_A = 25° C; unless otherwise specified)

PARAMETER	SYM.	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Operating Voltage	V_{DD}	-	2.4	-	5.5	V
Standby Current	I_{DD1}	No load, No Playing @5V	-	-	2	µA
Operating Current (Crystal Type)	I_{OP1}	No load	-	-	1	mA
Operating Current (Ring Type)	I_{OP2}	No load	-	-	1	mA
Input Low Voltage	V_{IL}	All Input Pins	V_{SS}	-	0.3V_{DD}	V
Input High Voltage	V_{IH}	All Input Pins	0.7V_{DD}	-	V_{DD}	V
Input Current for P0, P1,P2	I_{IN}	V_{DD} = 3V, V_{IN} = 0V	-	-	-6	µA
Input Current for /RESET	I_{IN1}	V_{DD} = 3V, V_{IN} = 0V	-	-	-6	µA
Output Current of P1, P2, P3	I_{OL}	V_{DD} = 3V, V_{OUT} = 0.4V	5	-	-	mA
	I_{OH}	V_{DD} = 3V, V_{OUT} = 2.7V	-3	-	-	mA
DAC (D/A full Scale)	I_{DAC}	V_{DD} = 4.5V, R_L = 100Ω	-4.0	-5.0	-6.0	mA
Pull-low Resistor	R_{PL}	TEST, OSCSEL Pins	100	-	-	KΩ

AC PARAMETERS

PARAMETER	SYM.	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Main-clock Frequency	FM	Ring type, ROSC = 1.2MΩ @3V,1.1MΩ@4.5V	2.7	3	3.3	MHz
Frequency Deviation by Voltage Drop for Ring Type Oscillator	$\frac{\Delta f}{f}$	$\frac{f(3V) - f(2.4V)}{f(3V)}$	-	-	10	%
Instruction Cycle Time	TINS	One machine cycle	1/12K	-	1/3K	S
POR Pulse Width	TPOR	-	1	-	-	mS

TYPICAL APPLICATION (for your reference only)

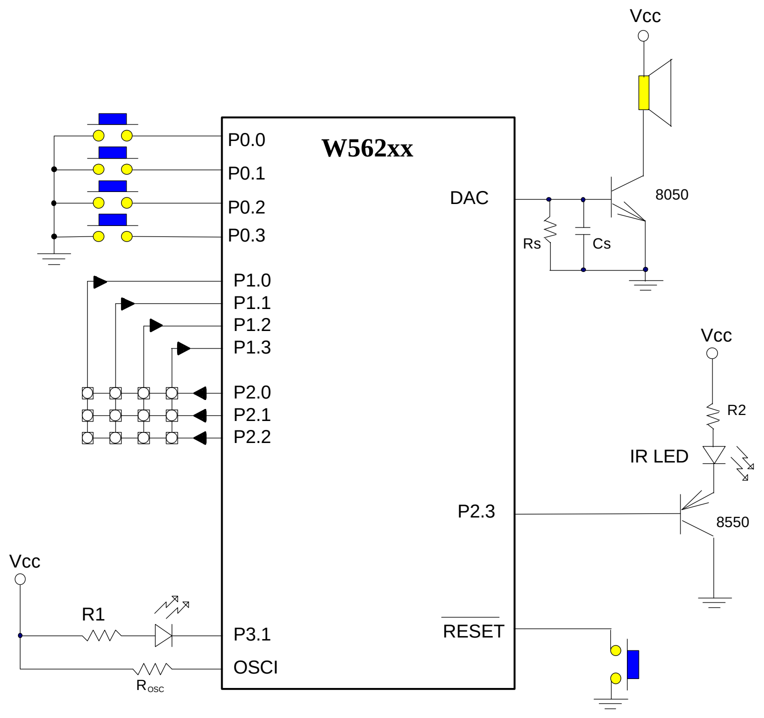


Figure 0-4. Application Circuit of W562xxx

ARCHITECTURE

This section discusses the internal architecture of W562xxx.
Following chart shows the major components of the W562xxx devices' internal architecture.

RAM and Registers

0x00 - 0x07	Working RAM
0x08 - 0x3F	General-purpose RAM
0x40 - 0x5F	Control Register
0x60 - 0xFF	ICE Register

The built-in RAM and registers map to the same address lines, which is called memory (RAM) mapped registers. The registers comprise three major parts: **1) control registers**, like ACC & IOR; **2) internal registers**, like SAR, CLP, STACK, and OPTION registers. They are used for ICE debugging purposes

The first 128 RAM-mapped locations are 4-bit wide and are suitable for data exchanges. The last 128 addresses are used for those registers that got more than 4 bits.

RAM

64 x 4 bits RAM, R0..R63, can be used to store data. They can only be addressed directly. The first eight locations, R0..R7, are used as **Working Register (WR),** denote as WR0 to WR7. They are useful in data moves from other RAM-mapped locations. For convenience, users can denote WR0 ~ WR7 as R0 ~ R7 to load data directly, and then rename to WR0 ~ WR7 for moving the data. The other data memories are used as general memory and can't operate directly with another RAM.

Example :

(1) Load data

```
LD R10, 1001b       ; the 4 bit value is loaded into the addressed location 10.
```

(2) Load & Move data

```
LD R2, 1001b        ; Working Register location 2 (WR2) is named as R2 for loading
MV R63, WR2         ; data into it , and then renamed to WR2 to move data to R63.
```

Control Registers

ACC

The ACC (Accumulator) is generally modified during MOVE and ALU operations to reflect the operation result. Branch instructions can be decided to be executed or not depending on the ACC content.

MODE

3	2	1	0
x	x	x	Timer clock

Default vaules upon power up are "0000", indicating clock source of timer is 32 Hz.

MODE.0	Clock Source	Overflow Period
0	32 Hz	30 mS - 8 S
1	32 KHz	30 uS - 8 mS

Example :

```
        LD MODE, 1101b          ;  bit1, 2, and 3 are don't care
                                ;  clock source of Timer is 32 Khz
```

IER

3	2	1	0
Timer	TG	Reserved	Reserved

0: Disable (Default)
1: Enable

If an interrupt source is disabled, the corresponding ISR (Interrupt Service Routine) won't be invoked upon the occurrence.

Example :

```
     LD IER, 1100b       ; Timer & Trigger interrupt are enable,
```

PER0 (Port Enable Register 0)

3	2	1	0
P0.3	P0.2	P0.1	P0.0

0: Disable (default)
1: Enable

The PER0 (Port Enable Register 0) defines the enable/disable condition for the falling-edge trigger of each P0 pin.

Example :

```
         LD PER0, 0011b         ; The pins P0.2 and P0.3 are disable.
```

PER1 (Port Enable Register 1)

3	2	1	0
P1.3	P1.2	P1.1	P1.0

0: Disable (default)
1: Enable

The PER1 (Port Enable Register 1) defines the enable/disable condition for the falling-edge trigger of each P1 pin when P1 is configured as input port.

Example :

```
        LD PER1, 1100b         ; Pins P1.0 and P1.1 are disable
```

P1 (Port 1 Register)

3	2	1	0
P1.3	P1.2	P1.1	P1.0

P1 is used for storing output values, provided that some pins of the port 1 is selected as outputs.

P2 (Port 2 Register)

3	2	1	0
P2.3	P2.2	P2.1	P2.0

P2 is used for storing output values, provided that some pins of the port 2 is selected as outputs.

P3 (Port 3 Register)

3	2	1	0
P3.3	P3.2	P3.1	P3.0

P3 is used for storing output values of the port 3.

IOR1 (I/O Register 1)

3	2	1	0
P1.3	P1.2	P1.1	P1.0

0: Input (Default)
1: Output
This register is used to specify the selection of input or output of P1.0 - P1.3 pins of port P1. A "0" selects input, while a "1" gives the output selection. Default value of IOR1 is 00h(after power on, or reset), which indicates the selection of all inputs, to avoid possible I/O conflicts with external circuits that may cause large current consumption.

Example :

```
        LD IOR1, 1100b        ; input pins : P1.0, P1.1
                             ; output pins : P1.2, P1.3
```

Once a pin was configured as input pin and Hi impedance (by set 0 in related bit of PCR1 or PCR2 register), this pin ***must not*** leave as floating to avoid large standby current.

IOR2 (I/O Register 2)

3	2	1	0
P2.3	P2.2	P2.1	P2.0

Same as IOR1 except for the selection of port P2, rather than port P1.
Example :

```
        LD IOR2, 1111b        ; port 2 is configured as output port
```

PCR1 (Port Configuration Register 1)

3	2	1	0
P1.3	P1.2	P1.1	P1.0

Defaults are "0" upon power up.

Together with IOR1, the state of port P1 can be configured as follows:

Pin Function	P1	PCR1	IOR1	Pin State
Input, High Z	x	0	0	High Z
Input, Internal Pull-up	x	1	0	Passive Pull-up
Output, Inverter	0	1	1	0
Output, Inverter	1	1	1	1
Output, Open Drain	0	0	1	0
Output, Open Drain	1	0	1	High Z

Example :

```
LD IOR1, 1111b        ; P1 is configured as an open-drain output port with
LD PCR1, 0000b        ; high  Z state
LD P1, 1111b
```

PCR2 (Port Configuration Register 2)

3	2	1	0
P2.3	P2.2	P2.1	P2.0

This register is used to configure the I/O type of port P2.

PSR (Processor Status Register)

PSR.3	PSR.2	PSR.1	PSR.0
BZ2	BZ1	Carry	Zero
Read only		Read / Write	Read only

0: False (Default upon power up)
1: True

BZ2 : busy or not of channel 2.

BZ1 : busy or not of channel 1.

Carry : carry flag is set or not.

Zero : zero flag is set or not (default is 1).

The PSR register needs to be stored first after an interrupt happens along with general-purpose ACC in order to prevent from possible changes made during ISR. They must be moved from temporarily stored RAM locations to PSR and ACC for restoring initial values correctly.

Example :

```
SETB PSR.1        ; set the carry flag (CF) to 1
```

TF0 (Trigger Flag of Port 0)

TF0 is used to latch the individual trigger event of port 0 (pins P0.0 - P0.3). TF0 register is organized as 4-bit binary register (TF0.0 to TF0.3). It can be read or cleared by "MV WRn , TF0", and "CLR TF0" instructions. The bit descriptions are as follows:

3	2	1	0
P0.3	P0.2	P0.1	P0.0

Bit 0 = 1 Falling-edge detected on P0.0
Bit 1 = 1 Falling-edge detected on P0.1
Bit 2 = 1 Falling-edge detected on P0.2
Bit 3 = 1 Falling-edge detected on P0.3

Default value is 0000B upon power up. Since each bit of TF0 will be set up to 1 whenever a falling-edge is detected on the corresponding pin (no debounce time). The TF0 must be cleared immediately after the ISR of dedicated TG interrupt is successfully invoked to prevent the undesired disturbance which may be caused by glitch on port 0 (see more detail in **Program Example** section).

Example :

```
        CLR   TF0              ; clear the TF0 register
        CLRB TF0.1             ; clear TF0 bit1
```

TF1 (Trigger Flag of Port 1)

3	2	1	0
P1.3	P1.2	P1.1	P1.0

Same as TF0 except for port P1, rather than port P0, when P1 is configured as trigger interrupt input.

TXF (Transmit enable flag)

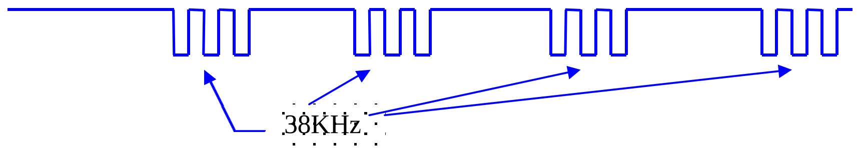
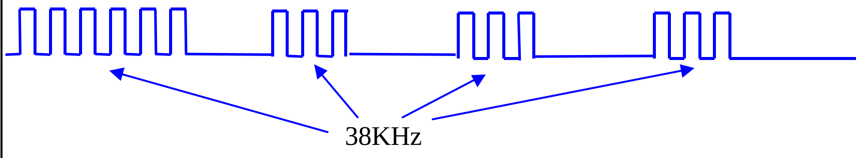
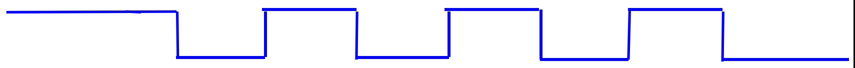
3	2	1	0
reserved	reserved	with low/high active	disable/enable 38KHz carrier

TXF.0 define IR carrier enable or not
= 0 disable IR 38KHz carrier
= 1 enable IR 38KHz carrier

TXF.1 define IR carrier with low or high active
= 0 P2.3 low active with 38KHz carrier
= 1 P2.3 high active with 38KHz carrier

default value (TXF.0, TXF.1) = (0, 0)

Control register v.s. P2.3 output waveform

program	P2.3 output waveform
SETB TXF.0 ; enable IR carry CLRB TXF.1 ; with low carrier CALL PATTERN	
SETB TXF.0 ; enable IR carry SETB TXF.1 ; with high carrier CALL PATTERN	
CLRB TXF.0 ; disable IR carry ;SETB TXF.1 ; don't care TXF.1 ; state, as the normal output pin CALL PATTERN	

```
PATTERN:                        CALL DELY2MS
        SETB P2.3               CLRB P2.3
        CALL DELY5MS            CALL DELY2MS
        CLRB P2.3               SETB P2.3
        CALL DELY2MS            CALL DELY2MS
        SETB P2.3               CLRB P2.3
        CALL DELY2MS            CALL DELY2MS
        CLRB P2.3               RTN
        CALL DELY2MS
        SETB P2.3
```

TIMER

7	6	5	4	3	2	1	0

This 8-bit timer downcounts every clock cycle, which is determined by the timer clock source. Upon overflow conditions that occur when the timer changes from 00h to FFh, the W56000 generates the timer INT to allow manipulation of timed events. It starts to downcount after the execution of "LD timer, operand". After the timer INT occurs under overflow conditions, the timer stops downcounting. Users have to re-start the timer operation by loading the timer with an appropriate value.

The timer interrupt happens every 30 uS - 8 second, depending on the selection of different clock sources, and can be disabled by IER.

Example :

```
        LD TIMER, 080h                ;  load  the period of timer into TIMER register
        ...............                ;  and the timer starts downcounting
        ...............
        TIMER_INTERRUPT :             ; timer interrupt entry
          ........
          LD TIMER, 080h              ; re-start the counting
          .........
         RTI                           ; return from interrupt
```

TEMPO

The tempo of W562xxx can not be dynamically controlled by register as well as if the ″Fix Beat Length″ of the Melody Converter was set then the tempo will be changed automatically.
The tempo of W562xxx as listing below:

bpm (beat per minute)

1091	**546**	**364**	**273**	**218**	**182**	**156**	**136**
121	**109**	**99**	**91**	**84**	**78**	**73**	**68**

Memory Partition

The W56000 divides the external memory space into several different partitions for the purpose of dual-channel operation along with the program execution, they are: Option, IV (Interrupt Vector), COLA (Channel Operation List Area), Control Program, and ADPCM/note.

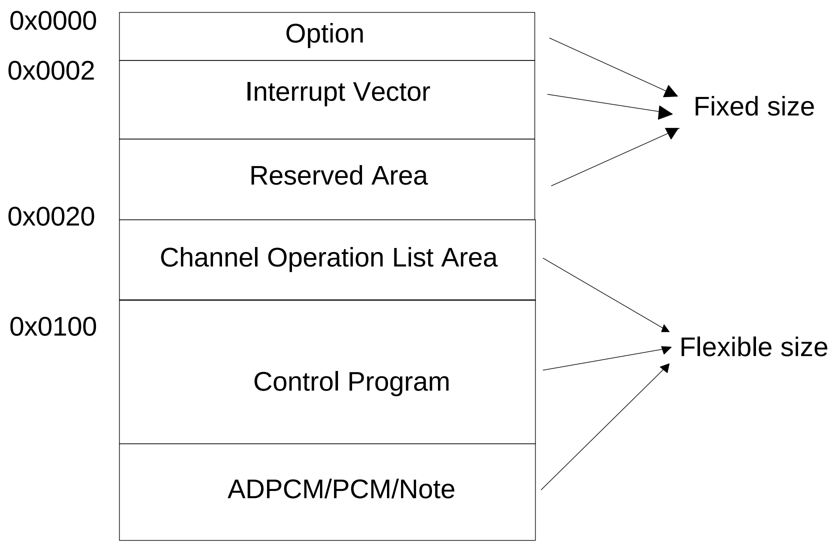


Figure 0-5. Memory Partition

Operation Flow

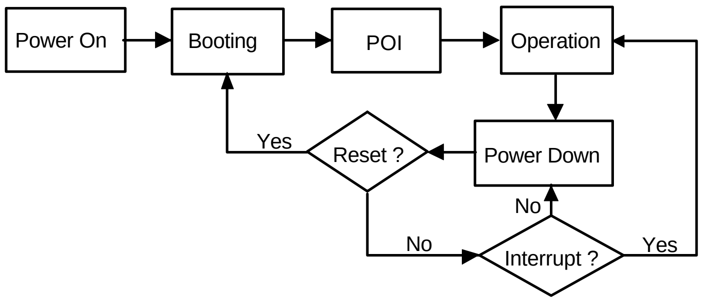


Figure 0-6. Operation Flow Chat

The Figure 0-6. shows the operation flow chart of W56xxx device.

The W562xxx enters power down (or called standby) mode if and only if the following conditions are met:

1) END instruction executed;
2) Dual-channel operations cease;
3) Timer disabled or not activated since last overflow.

After entering the power down mode, only /RESET and TG interrupts can wake up the W56xxx. Followings are two typical procedures to get in/out of the power down mode.

1) Power on => Boot (Option < 200 mS) => POI => Operation => Power down.
2) Power down mode => Wake up (RESET, TG interrupt) => Operation => Power down.

Interrupt

Keyword	Name	Address	Instruction
POI:	POR/Reset	002H	JP POI
Reserved	Reserved	008H	Reserved
TG:	TG	00AH	JP TG
Timer_interrupt:	Timer	00CH	JP Timer_interrupt

Table 0-1. Interrupt vector

When an ISR (Interrupt Service Routine) is in service, others (TG, Timer) are disabled by H/W automatically. Only if otherwise enabled by S/W program through instructions that modify content of IER, like "LD IER, operand", (the disable condition be released by this instruction upon the updating of IER), or enable interrupt EN INT , other interrupts (except for POR) are disabled until the reaching of RTI (interrupt enabled again by H/W automatically), which indicates the finish of an ISR. Special care goes to the prevention of STACK overflow, since only maximum 8-level stack register is shared among all interrupt sources, and CALL instruction.

The POR/Reset is Non-Maskable Interrupt (NMI). The program execution and channel operations all return to initial states as in power up conditions.

Other interrupt sources are processed in a polling sequence of the priority: TG => Timer, if occur simultaneously. If an interrupt is disabled during ISR of other interrupts, it is latched and shall be serviced right after the release of the disable condition.

Trigger

Since all P0 & P1 triggers share the same TG interrupt vector, users have to further differentiate the real one from others by programming. Also, trigger debounce should be considered in the TG section. Though a little bit complex than PowerSpeech, which is done by H/W debounce circuit, but on the other hand, it do mean more flexibility.

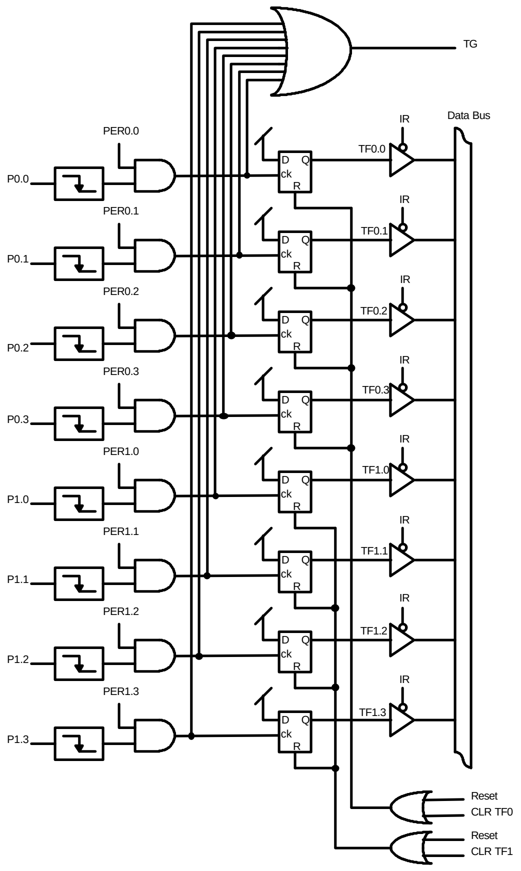


Figure 0-7. Trigger Scheme

Figure 0-7 shows the structure of trigger scheme. Two program examples based on this structure is listed below to show how to configure the ISR for trigger interrupt.

Example (1) : Use port 0 as interrupt source

```
TG:                                 ; TG is keyword.
    MV WR0, P0
    CJE R0, 1111B, BACK
    MV WR1, TF0
    CJE R1, 0000B, BACK
    call debounce                   ; Software debounce used to prevent  the glitch on port 0
    XOR P0, WR0
    JZ SCAN0
```

```
BACK:
     CLR TF0
     RTI

SCAN0:                              ; differentiate which pin of port 0 is the real one trigger source.
     MV WR0, TF0
     MV ACC, WR0
     CLR TF0
     JB0 P00
     JB1 P01
     JB2 P02
     JB3 P03
     RTI

P00:                                       ; ISR for trigger source P0.0
    PLAY CH1, H4+some_2+T4
    RTI

P01:
    PLAY CH1, H4+arround_2+T4          ; ISR for trigger source P0.1
    RTI

P02:                                       ; ISR for trigger source P0.2
    PLAY CH1, H4+feel_2+T4
    RTI

P03:
    PLAY CH1, H4+dance_2+T4              ; ISR for trigger source P0.3
    RTI

debounce:                                  ; soft ware debounce,  a delay time of about
       LD R11, 0111b                       ; 512 instructions cycle time = 40 mS
a:
       LD R12, 1111b
b:
        DJNZ R12, b
        DJNZ R11, a
       RTN
```

Example (2) : Use both port 0 and port 1 as trigger source

```
TG:                                 ; TG is keyword.
     MV WR0, P0
     CJE R0, 1111B, TRIG1
     MV WR1, TF0
     CJE R1, 0000B, TRIG1
     call debounce
     XOR P0, WR0
     JZ SCAN0
TRIG1:                                   ; software debounce for port 1
     CLR TF0
     MV WR0, P1
     CJE R0, 1111B, BACK
     MV WR1, TF1
     CJE R1, 0000B, BACK
```

```
    call debounce
    XOR P1, WR0
    JZ SCAN1
BACK:
    CLR TF1
    RTI

SCAN0:
    MV WR0, TF0
    MV ACC, WR0
    CLR TF0
    JB0 P00
    JB1 P01
    JB2 P02
    JB3 P03
    RTI

SCAN1:                        ; differentiate which pin of port 1 is the real one trigger source
    MV WR0, TF1
    MV ACC, WR0
    CLR TF1
    JB0 P10
    JB1 P11
    JB2 P12
    JB3 P13
    RTI
```

Timer

The timer of the W56xxx is a quite simple mechanism to facilitate applications with timed events. Once the timer interrupt is enabled, the timer starts to down-count as long as the "LD timer, operand" is executed. Upon reaching the overflow condition, the timer interrupt occurs and the timer_interrupt subroutine is serviced.

Users have to re-load the timer with a new value in order to start the timer again, since there's no auto-reload function in it.

Interrupt Control

During the period of a specific ISR, other interrupts except for POR & synth are disabled by H/W automatically. Users may enable certain interrupts under S/W control: "LD IER, operand", or "EN INT".

Under normal conditions, the disabled interrupt situation disappears automatically when RTI of that specific ISR is encountered. Figure 0-8 shows the arbitrator of all the interrupt sources.

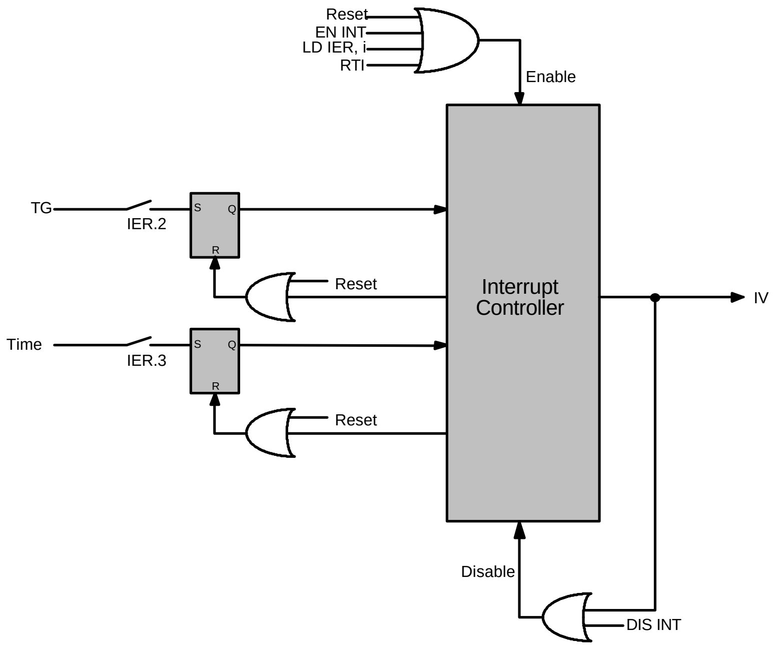


Figure 0-8. Interrupt Arbitrator

Speech Synthesis

The W562xxx offers two ADPCM synthesizers, synth1 and synth2, for voice reproduction. The operation of the synthesizers are the same as Winbond's PowerSpeech devices, and are description as below.

Channel Operation List (COL)

A Channel Operation (CO), which could be a voice segment, certain period of silence, or a melody phrase, is used to indicate what a channel is to playback with. A COL (Channel Operation List) contains a list of as many CO's as customers' needs for a certain channel, channel 1 or channel 2. The following example shows two COLs for channel 1 and channel 2, respectively.

Ch1: H4 + V1 + Song1 + Song2 + V2 + T4
Ch2: H4 + V1 + V2 + [silence] + V3 + T4

Where H4, T4 are the head file and tail file of ADPCM/melody voice data. And V1, V2, V3 are voice segments (*.wam or *.src); [silence] is certain period of silence; Song1 and Song2 are melody phases (*.dm).

There are many head and tail files for your option: H4/T4, H41/T41, H42/T42, and H43/T43 which can make sound smoothly in order to prevent the "pop" sound.

The silence length is represented by 2's complement of the actual length in unit of $(3khz)^{-1}$ due to the essence of up counting, instead of down counting. The W562xxx is said to finish the counting of the silence length, if overflow condition occurs. Since the width of the register that holds the silence length is 16 bit, the maximum silence length that can be implemented with each CO is 64

* 1024 * (3khz)$^{-1}$ = 21.8 second. Of course, if customer really wants longer silence length, just add more CO's.
For example :

PLAY CH1, h4+[0BB8]+t4 ;denote the channel is playing a silence with a period of 1 second.

Section Control

The W562xxx provides various sampling frequencies for ADPCM speech synthesis which can be set by the section control function in the channel operation.

The variable frequency which is provided for ADPCM synthesis are listed below:

Option	SR
	ADPCM
0	4.8 KHz
1	6 KHz
2	8 KHz
3	12 KHz

Table 0-2. Sampling frequency

The syntax of section control is listed below :

$$\text{PLAY CH1/ CH2, h4 + voice}\underset{\text{section control}}{\_}\underset{\text{sampling rate}}{s}\text{ + t4}$$

Three examples are given to show how the section control works.

Example (1) :

PLAY CH1, h4 + V1_ 3+t4 ; the voice V1 is played back with a sampling rate of
; 12 KHz

Example (2) :

PLAY CH1, h4 + V1_ 2+t4 ; the voice V1 is played back with the sampling rate
; of 8 KHz

Dual Tone Melody Generation

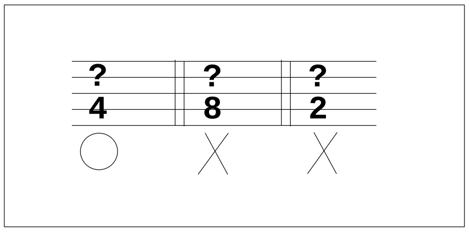
W562xxx provides the D/A output bit selection: 6/ 7/ 8 bits option.
The max. volume is 8 and 6 is the min. volume. While W562xxx is only playing the melody the 8 bits will make the volume loud. If melody just as the background music, the 6 bits will make the music more tender.
It is controlled like following example:

```
play ch1,h4+melody_xx6+t4
play ch1,h4+melody_xx7+t4
play ch1,h4+melody_xx8+t4
```

xx: means don't care.

There are some concerned points when we are handling W562xxx dual tone melody:
1) Pitch window: from G3# to C7
2) Every note's beat length must be more than 1/4 beat.

3) If you selected the "fix beat length" all the notes will be reserved, even less than 1/4 beat, but it will occupy more ROM size.
4) If you heared a "pop" sound at the end of the melody. The last note of the melody must take longer duration to prevent the "pop" sound. W562xxx need 2.5 sec to decay the notes volume to DC level.
5) track1 and track2 can't used the same pitch at the same time. The notes will disappear or have the smaller volume.
6) NH4/NT4 will make W562xxx error, please used H4/T4, H41/T41, H42/T42, or H43/T43.
7) Anytime of track1 and track2 only have one note. Else, converter will select one of notes from track1/track2 automatically to make *.dm file.
8)



Following above rules, W562xxx will play the complete and perfect dual tone melody.

Basic Structure of the W56xxx Program

A program is usually divided into 4 areas: Body Declaration, Constant Area, Macro Area, and Command Area. It may contain a structure as below:

Body Body Declaration	Body declares here
CONSTANT Constant Area	Constant area starts here
MACRO Macro Area	Macro area starts here
CODE Command Area	Command area starts here

Body Declaration

Body has to declare first in the program to acknowledge the IC body type and ROM size used. Missing body declaration will cause a compilation error. The body type supported now are listed below.

BODY	W562S08	W562S10	W562S12	W562S15	W562S20	W562S25	W562S30
	W562S40	W562S50	W562S60	W562S80	W562S99	W562M02	-

Note: Body release is subject to change without prior notice. For exact information, please check with Winbond's sales representatives.

Constant Area

```
SYNTAX
  DEFINE  <Constant Name>  <Value>
```

<Constant Name>

A constant name is a user-defined symbol. Refer to 0 for syntax of symbol.

<Value>

Value can be of different types, register, number, channel, clp, ram or wr. Please refer to section 0 for type information.

The Assembler, basically, substitutes the <Constant Name> with the <Value> while it parses through Command Area.

NOTE: The assembler is not case-sensitive.

For example:

```
Define   BUFFER1    32H    ;÷ Define value of Buffer 1 as immediate value 32H.
Define   BUFFER2    R13    ;÷ Define value of Buffer 2 as the 13th  RAM.
Define   BUFFER3    WR3    ;÷ Define value of Buffer 3 as the 3rd WR.
Define   FLAG1      1011B  ;÷ Define value of FLAG 2 as immediate value 1011B.
```

Macro Area

The Macro Area may consist of several macro declarations or none. After the macro has been defined, the macro name can be called in the program. While compiling, the assembler will look for the macro name, and expand it to the commands it defines. The syntax of macro declaration is described.

```
SYNTAX
  MACRO  <Macro Name>  <Arguments>
         <Commands>
  ENDM
```

<Macro Name>

A macro name is a user-defined symbol. Refer to section 0 for the syntax of symbol.

<Arguments>

Macro may have no arguments, or several arguments. Commands are used to separate between arguments. An argument is a user-defined symbol. The maximum number of arguments are 20.

<Commands>

Macro should contain at least one command. Labels and directives are not allowed in macro, and macro can not be nested. In short, a macro command can not be a macro declaration. However, a macro command can use arguments, which will then be substituted by the parameters of a macro call, for operands.

For example:

```
 MACRO  ADD2  arg1, arg2    ;÷ Define the macro name as ADD2.
                            ;÷ Define 2 arguments, arg1 and arg2.
           ADD arg1, arg2   ;÷ Define macro command ADD.
ENDM
```

Command Area

The Command Area may consist of several commands and/or directives. The syntax of a command is described as follows

SYNTAX

<Label> <Mnemonic> <Operand> <Comment>
<Label> <Directive>

<Label>

A label is a user-defined symbol that is followed by a colon. It is not a mandatory part of the program. The label is used to name a program location, and to label entry, skipping and branching of the program. It can be used to change the order of program execution by using commands, such as JP and CALL, to jump to locations or call routines. After the program is compiled, all labels are converted into absolute address values. Program labels make it easy to find and remember program locations without having to compute the memory address. It is convenient for modifying a program.

<Mnemonic>

Mnemonics are reserved names for instruction op codes.

<Operand>

<Operand> is optional. Operands provide the data for mnemonics to act on. There may be from zero to three operands, depending on the op code, and they are separated by commas. Operands take the form of either literals or symbols for data items. Symbols are assumed to be assigned to data items declared in the Constant Area.

Operands can be of the following types:

1. Symbol

An symbol is an alpha-numeric string (0~9, a   z, A   Z, and _) that starts with an alphabetic character and is allowed a maximum length of 40 characters. It must not be a keyword, or a predefined symbol name.

2. Number

Numbers, or immediate values, accept four forms:

<u>Binary:</u>
Binary numbers are base 2 numbers, and are represented by a string of binary digits followed by the character B. A binary digit is a character from {0, 1}. For example, 0101B is equivalent to the decimal number 9.

<u>Octal</u>:
Octal numbers are base 8 numbers, and are represented by a string of octal digits followed by the character O. An octal digit is a character from {0-7}. For example, 11O is equivalent to the decimal number 8.

<u>Decimal:</u>
Decimal numbers are base 10 numbers, and are represented by a string of decimal digits. A decimal digit is a character from {0-9}.

<u>Hexadecimal:</u>
Hexadecimal numbers are base 16 numbers, and are represented by a string of hexadecimal digits followed by the character H. A hexadecimal digit is a character from {0-9, a-f, A-F}. A leading zero should be added if the hexadecimal number begins with one of digits {a-f, A-F}. For example, 0AH is equivalent to the decimal number 15.

3. RAM

RAMs begin with a character R, and followed by a decimal number from 0 to 63. It is used to store data, and move data from/to any other RAM-mapped locations. The first eight locations of RAM, also called Working Registers (WR). RAMs can only be addressed directly. For example, R10 is the location 10 in RAM.

4. Working Register

Working Registers begin with characters WR, and followed by a decimal number from 0 to 7. It is actually RAM mapped locations from 0 to 7. Working Registers can only be addressed directly. For example, WR0 is the location 0 in RAM.

5. Register

Register names are reserved names. Refer to for more details. A list of registers is provided.

REGISTER	DESCRIPTION
ACC	Accumulator
MODE	Mode Register
IER	Interrupt Enable Register
PER0	Port Enable Register 0
PER1	Port Enable Register 1
P0	Port 0 Register
P1	Port 1 Register
P2	Port 2 Register
P3	Port 3 Register
IOR1	I/O Register 1
IOR2	I/O Register 2
PCR1	Pin Configuration Register 1
PCR2	Pin Configuration Register 2
VOL1	Volume 1 Register
VOL2	Volume 2 Register
PSR	Processor Status Register
TIMER	Timer Register
TXF	Transmit enable flag

6. Channel

Channels are used to identify which channel is used for voice output. Channels start with characters CH, and followed by number 1 or 2. For example: PLAY CH1, RING is to output a voice RING to channel 1.

7. Channel List Pointer

Channel List pointers are used to identify which channel list pointer is used. Channel List pointers start with characters CLP, and followed by number 1 or 2.

<Comment>

A good comment is a basic part in writing a program. The comments allow users to give the necessary explanations of commands. A comment starts with a semi-colon, and followed by any characters. The assembler will ignore a comment till end of the line.

<Directive>

Directives are also called pseudo instructions. The W56xxx Assembler provides several directives:

1. Define frequency

This directive is used to define the sample frequency.

SYNTAX

FREQ n<Frequency>

<Frequency>

It is the sample frequency of voice output, n= 0 to 3.

For example:

```
FREQ  3          ; Define the sample frequency of voice output as 3.
```

2. Play voices

This directive is used to play voices. It actually will expand into 4 commands:

```
STOP <channel>
LD <clp>, <voice list name>
RD <clp>
PLAY <channel>
```

SYNTAX

PLAY <Channel> , <Voice List>

<Channel>

Specify which channel to be output to. Only Ch1 and Ch2 are valid.

<Voice List>

Voice list may contain voice files, and/or length of silence for channel output. A voice file can be of two types: ADPCM (*.wam), or dual tone Melody(*.dm). The extension of voice file is not needed, the Assembler will automatically look for the voice files available in the local directory. A list of voices are separated by a +.

For example:

```
PLAY ch1, H4+Happy+[15]+Birthday+T4 ; Output 4 voice files, H4, Happy, Birthday, and T4,
                                    ; and a silence length 15h.
```

Furthermore, when the voice list contains a large number of voice files , 50 files or more, and can't be placed in one line in text editor, the voice list can be separated into several lines and connected by the notation "\". The assembler will look it as a single list.

For example:

```
PLAY ch1, H4+Happy+[15]+Birthday+bird  \
           +cat+dog+A+B+  \
            C+T4
```

are the same as

```
PLAY ch1, H4+Happy+[15]+Birthday+bird+cat+dog+A+B+C+T4
```

3. Declare speech

This directive is used to declare the type of speech files. From Speech Coding System Ver 7.54, ADPCM speech is the default speech file. All legal file type, for example, PCM files and WAV files, will be automatically converted to ADPCM files before compiling into .OBJ file. However, explicitly declaring speech file type, will retain the file type. Explicitly declaration will ensure the correct speech files will be used.

SYNTAX

SRC/WAM/DM/TM <filename> { , <filename> }

<filename>

It is the speech filename without extention name.

For example:

```
    WAM  dog                        ; define the ADPCM speech file. (default)
     DM    singing                   ; define the dual Tone melody file.
```

Winbond Electronics Corp.

INSTRUCTION SET

Instruction Set List

Mnemonics	Operation	Flag Affected	CYCLE
Arithmetic			
ADD Rn, i	ACC, Rn <-- Rn + i	Zero/Carry	1
ADDC Rn, i	ACC, Rn <-- Rn + i + carry	Zero/Carry	1
ADD Rn, WR	ACC, Rn <-- Rn + WR	Zero/Carry	1
ADDC Rn, WR	ACC, Rn <-- Rn + WR + carry	Zero/Carry	1
SUB Rn, i	ACC, Rn <-- Rn - i	Zero/Carry	1
SUBC Rn, i	ACC, Rn <-- Rn - i - carry	Zero/Carry	1
SUB Rn, WR	ACC, Rn <-- Rn - WR	Zero/Carry	1
SUBC Rn, WR	ACC, Rn <-- Rn - WR - carry	Zero/Carry	1
INC Rn	ACC, Rn <-- Rn + 1	Zero/Carry	1
DEC Rn	ACC, Rn <-- Rn - 1	Zero/Carry	1
SETB Rn.b	ACC, Rn <-- Rn | (2^b) ; b = 0 ~ 3	Zero	1
CLRB Rn.b	ACC, Rn <-- reg (Rn) & (1's complement of 2^b);b = 0 ~ 3	Zero	1
Logic Operations			
AND Rn, i	ACC, Rn <-- Rn & i	Zero	1
AND Rn, WR	ACC, Rn <-- Rn & WR	Zero	1
OR Rn, i	ACC, Rn <-- Rn | i	Zero	1
OR Rn, WR	ACC, Rn <-- Rn | WR	Zero	1
XOR Rn, i	ACC, Rn <-- Rn ^ i	Zero	1
XOR Rn, WR	ACC, Rn <-- Rn ^ WR	Zero	1
NOT Rn	ACC,Rn <-- 1's complement of Rn	Zero	1
Shift & Rotate			
RORC Rn	ACC.b, Rn.b <-- Rn.(b+1); ACC.3, Rn. 3 <-- carry, carry <-- Rn.0	Carry	1
ROLC Rn	ACC.b, Rn.b <-- Rn.(b-1),; ACC.0, Rn. 0 <-- carry, carry <-- Rn.3	Carry	1
SHRC Rn	ACC.b, Rn.b <-- Rn.(b+1), ACC.3, Rn. 3 <-- 0, carry <-- Rn.0	Carry	1
SHLC Rn	ACC.b, Rn.b <-- Rn.(b-1), ACC.0, Rn.	Carry	1

	0 <-- 0, carry <-- Rn.3		
Data Move			
LD Rn, i	Rn <-- i, Rn : RAM mapped register (0 ~ 127) i : immediate value, 4 bit.	-	1
LD Rn, k	Rn <-- k, Rn: RAM mapped register (128 ~164) k : immediate value, 8 bit	-	1
LDR Rn, i	Rn $\leftarrow$ $v_3v_2v_1v_0$,, v_j= r * b_j, r : random number (1 or 0), b_j = 1 for random, j = 0~3,	-	1
MV Rn, WR	Rn <-- WR	-	1
MV WR, Rn	WR <-- Rn	-	1
Branch			
JP label	PC <-- label	-	2
JB0 label	PC <-- label, if ACC.0 = 1; PC <-- PC++, if not	-	2
JB1 label	PC <-- label, if ACC.1 = 1; PC <-- PC++, if not	-	2
JB2 label	PC <-- label, if ACC.2 = 1; PC <-- PC++, if not	-	2
JB3 label	PC <-- label, if ACC.3 = 1; PC <-- PC++, if not	-	2
JZ label	PC <-- label, if PSR.0 = 1; PC <-- PC++, if not	-	2
JNZ label	PC <-- label, if PSR.0 = 0; PC <-- PC++, if not	-	2
JC label	PC <-- label, if carry flag = 1; PC <-- PC++, if not	-	2
JNC label	PC <-- label, if carry flag = 0; PC <-- PC++, if not	-	2
JBZ1 label	PC <-- label, if Busy1 flag = 1; PC <-- PC++, if not	-	2
JBZ2 label	PC <-- label, if Busy2 flag = 1; PC <-- PC++, if not	-	2

CJNE Rn, i,label	ACC <-- Rn - i; PC <-- label, if ACC ! = 0; otherwise
	PC <-- PC++
		Zero/Carry	2

CJE Rn, i,label	ACC <-- Rn - i; PC <-- label, if ACC = 0; otherwise
	PC <-- PC++
		Zero/Carry	2

CJNE Rn,WR,label	ACC <-- Rn - WR; PC <-- label, if ACC ! = 0 ; otherwise PC <-- PC++	Zero/Carry	2
CJE Rn,WR,label	ACC <-- Rn - WR; PC <-- label, if ACC = 0; otherwise PC <-- PC++	Zero/Carry	2
DJNZ Rn,label	ACC, Rn <-- Rn - 1; PC <-- label, if ACC ! = 0; otherwise PC <-- PC++	Zero/Carry	2
DJZ Rn, label	ACC, Rn <-- Rn - 1; PC <-- label, if ACC = 0;	Zero/Carry	2

	otherwise PC <-- PC++		

Mnemonics	Operation	Flag Affected	CYCLE
Subroutine			
CALL label	STACK <-- PC+1, then PC <-- label.	-	2
RTN	PC <- STACK, used in call subroutine return	-	1
RTI	PC <- STACK, used in interrupt return	-	1
Others			
NOP	No operation	-	1
END	Program execution stops	-	1
EN INT	Enable interrupt source	-	1
DIS INT	Disable interrupt source	-	1
PLAY CH1/2, h4+voice+t4	Play voice segment used channel 1/2	BZ1/BZ2	3
STOP CH1/2	Stop playing channel 1/2	BZ1/BZ2	1
CLR TF0/1	TF0/1 <-- 0000B	-	1

Legend

Rn: 0 ~ 63, including WR0 ~ WR7, control registers, internal registers

WR: Working Register, WR0 ~ WR7

PC: Program Counter

i: immediate value, 4 bit

k: immediate value, 8 bit

label: 0 ~ 65535 word

Operator

! =: Not equal

&: AND

|: OR

^: XOR

<--: Data transfer

NOTE: ACC, Carry flag and Zero flag are modified implicitely after ALU operations.

Instruction Description

Arithmetic

Instruction Set	Description	Example
ADD Rn, i	*ACC, Rn ‹ Rn + i* Add an immediate data i to register Rn. The result is kept in "Rn" and ACC. Flag affected: CF, ZF[1]	**ADD R10, 0011b** Memory. exec. Before exec. After i 0011b 0011b R10 1100b 1111b ACC --- 1111b
ADDC Rn, i	*ACC, Rn ‹ Rn +i + CF* Add an immediate data i and CF to register Rn. The result will be kept in Rn and ACC. Flag affected: CF, ZF	**ADDC R10, 0011b** Memory. exec. Before exec. After i 0011b 0011b CF 1 1 R10 1100b 0000b ACC --- 0000b
ADD Rn, WR	*ACC, Rn ‹ Rn +WR* Add the content of WR to register Rn. The result will be kept in Rn and ACC. Flag affected: CF, ZF	**ADD R10, WR1** Memory. exec. Before exec. After WR1 1011b 1011b R10 1100b 0111b ACC --- 0111b
ADDC Rn, WR	*ACC, Rn ‹ Rn+WR + CF* Add the content of WR and CF to register Rn. The result will be kept in Rn and ACC. Flag affected: CF, ZF	**ADDC R10, WR1** Memory. exec. Before exec. After WR1 1011b 1011b CF 1 1 R10 1100b 1000b ACC --- 1000b
SUB Rn, i	*ACC, Rn ‹ Rn - i* Subtract an immediate data i from the register Rn. The result will be kept in Rn and ACC. Flag affected: CF, ZF	**SUB R11, 0101b** Memory. exec. Before exec. After i 0101b 0101b R11 1101b 1000b ACC --- 1000b

[1] **CF** denotes **C**arry **F**lag and **ZF** denotes **Z**ero **F**lag

SUBC Rn, i	*ACC, Rn ‹ Rn - i - CF* Subtract an immediate data i and CF from the register Rn. The result will be kept in Rn and ACC. Flag affected: CF, ZF	**SUBC R11, 0101b** Memory. exec. | Before exec. | After i | 0101b | 0101b CF | 1 | 0 R11 | 1101b | 0111b ACC | --- | 0111b
SUB Rn, WR	*ACC, Rn ‹ Rn - WR* Subtract the content of WR from the register Rn. The result will be kept in Rn and ACC. Flag affected: CF, ZF	**SUB R11, WR0** Memory. exec. | Before exec. | After WR0 | 0101b | 0101b R11 | 1101b | 1000b ACC | --- | 1000b
SUBC Rn, WR	*ACC, Rn ‹ Rn - WR - CF* Subtract the content of WR and CF from the register Rn. The result will be kept in Rn and ACC. Flag affected: CF, ZF	**SUBC R11, WR0** Memory. exec. | Before exec. | After WR0 | 0101b | 0101b CF | 1 | 0 R11 | 1101b | 0111b ACC | --- | 0111b
INC Rn	*ACC, Rn ‹ Rn + 1* Increment register Rn by 1. Flag affected: ZF, CF	**INC R12** Memory. exec. | Before exec. | After R12 | 5 | 6 ACC | --- | 6
DEC Rn	*ACC, reg ‹ Rn - 1* Decrement register Rn by 1. Flag affected: ZF, CF	**DEC R12** Memory. exec. | Before exec. | After R12 | 5 | 4 ACC | --- | 4
SETB Rn.b	*AC, Rn ‹ Rn | (2^b), b = 0~3* Set one of the register bits to 1 without altering the values of other bits. Flag affected: ZF	**SETB MODE.3** Memory. exec. | Before exec. | After MODE | 0000b | 1000b ACC | --- | 1000b

CLRB Rn.b	<u>*ACC, Rn ‹ Rn & (1's complement of 2^b) b = 0~3*</u> Set one of the register bits to 0 without altering the values of other bits. <u>Flag affected:</u> ZF	**CLRB IER.3** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> IER 1100b 0100b ACC --- 0100b

Logic Operations

AND Rn, i	<u>*ACC, Rn ‹ Rn & i*</u> Bitwise AND operation of register Rn and an immediate value i. The result will be kept in Rn and ACC. <u>Flag affected:</u> ZF	**AND R2, 0010b** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> R2 0110b 0010b i 0010b 0010b ACC --- 0010b
AND Rn, WR	<u>*ACC, Rn ‹ Rn & WR*</u> Bitwise AND operation of register Rn and WR. The result will be kept in Rn and ACC. <u>Flag affected:</u> ZF	**AND R2, WR3** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> R2 1010b 1000b WR3 1100b 1100b ACC --- 1000b
OR Rn, i	<u>*ACC, Rn ‹ Rn | i*</u> Bitwise OR operation of register Rn and an immediate value i. The result will be kept in Rn and ACC. <u>Flag affected:</u> ZF	**OR R0, 0011b** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> R2 1010b 1011b i 0011b 0011b ACC --- 1011b

OR Rn, WR	*ACC, Rn ‹ Rn | WR* Bitwise OR operation of register Rn and WR. The result will be kept in Rn and ACC. Flag affected: ZF	**OR R0, WR2** Memory. exec. Before exec. After R2 1010b 1011b WR2 0011b 0011b ACC --- 1011b
XOR Rn, i	*ACC, Rn ‹ Rn ^ i* Bitwise XOR operation of register Rn and an immediate value i. The result will be kept in Rn and ACC. Flag affected: ZF	**XOR R7, 0101b** Memory. exec. Before exec. After R7 1101b 1000b i 0101b 0101b ACC --- 1000b
XOR Rn, WR	*ACC, Rn ‹ Rn ^ WR* Bitwise XOR operation of register Rn and WR. The result will be kept in Rn and ACC. Flag affected: ZF	**XOR R7, WR5** Memory. exec. Before exec. After R7 1101b 1000b WR5 0101b 0101b ACC --- 1000b
NOT Rn	*ACC, Rn ‹ 1's complement of Rn* Performs 1's complement of the register Rn. The result is kept in Rn and ACC. Flag affected: ZF	**NOT R7** Memory. exec. Before exec. After R7 0010b 1101b ACC --- 1101b

Shift and Rotate

RORC Rn	*ACC.b, Rn.b ‹ Rn.(b+1);* *ACC.3, Rn.3 ‹ CF; CF ‹ Rn.0* The content of register Rn is rotated right one bit, bit 0 is rotated into CF, and CF is rotated into bit 3 of Rn. The result is kept in Rn and ACC. Flag affected: CF	**RORC R16** Memory. exec. Before exec. After R16 0010b 1001b CF 1 0 ACC --- 1001b

ROLC Rn	*ACC.b, Rn.b ‹ Rn.(b -1);* *ACC.0, Rn.0 ‹ CF; CF ‹ Rn.3* The content of register Rn is rotated left one bit, bit 3 is rotated into CF, and CF is rotated into bit 0 of Rn. The result is kept in Rn and ACC. Flag affected: CF	**ROLC R16** Memory. Before exec. After exec. R16 0010b 0101b CF 1 0 ACC --- 0101b
SHRC Rn	*ACC.b, Rn.b ‹ Rn.(b +1);* *ACC.3, Rn.3 ‹ 0; CF ‹ Rn.0* The content of register Rn is shifted right one bit, bit 0 is shifted into CF, and bit 3 of Rn is replaced with "0". The result is kept in Rn and ACC. Flag affected: CF	**SHRC R16** Memory. Before exec. After exec. R16 0011b 0001b CF 0 1 ACC --- 0001b
SHLC Rn	*ACC.b, Rn.b ‹ Rn.(b - 1);* *ACC.0, Rn.0 ‹ 0; CF ‹ Rn.3* The content of register Rn is shifted left one bit, bit 3 is shifted into CF, and bit 0 of Rn is replaced with "0". The result is kept in Rn and ACC. Flag affected: CF	**SHLC R16** Memory. Before exec. After exec. R16 1011b 0110b CF 0 1 ACC --- 0110b

Data Move

Instruction Set	Description	Example
LD Rn, i	*Rn ‹ i* Load register Rn with an immediate 4 bit value i. Flag affected: none	**LD IER, 1100** Memory. Before exec. After exec. i 1100b 1100b IER --- 1100b

LD Rn, k ***** For internal register only (address 128 ~ 164)	*Rn ‹ k* Load internal register Rn (address 128 ~ 164) with an immediate 8 bit value k. Flag affected: none	**LD TIMER, 0A9h** Memory. exec. / Before exec. / After k / 0A9h / 0A9h TIMER / --- / 0A9h
LDR Rn, i	*Rn ‹ $v_3v_2v_1v_0$,, v_j= r * b_j, r : random number (1 or 0), b_j = 1 for random, j = 0~3.* Load register Rn with a random number, which is determined by the following formula : i = $b_3b_2b_1b_0$ (binary form) Rn.j = r · b_j, $0 \le j \le 3$ where Rn.j = bit j of register Rn in binary representation, r = random bit(0 or 1), which is generated by H/W, b_j = bit j of "i" (4 bit, 0 - 15) in binary representation. Flag affected: none	**LDR R2, 7** Memory. exec. / Before exec. / After i / 0111b / 0111b R2 / --- / 0rrrb

MV Rn, WR[*]	*Rn ‹ WR* Move the content of WR to Rn. Flag affected: none	**MV R12, WR0** Memory. exec. / Before exec. / After WR0 / 1110b / 1110b R12 / --- / 1110b
MV WR, Rn	*WR ‹ Rn* Move the content of Rn to WR. Flag affected: none	**MV WR2, R10** Memory. exec. / Before exec. / After R10 / 1110b / 1110b WR2 / --- / 1110b

Branch

Instruction Set	Description	Example

[*] Data move can only be accessed between general registers (Rn) and Working Register (WR0 ~ WR7). See the Program Example for more details.

JP label	<u>*PC ‹ label*</u> The PC is replaced with "label", and an unconditional branch occurs. <u>Flag affected:</u> none	**JP EXIT** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> EXIT 1A9h 1A9h PC --- 1A9h (In this example, the "EXIT" label is compiled to 1A9h.)
JB0 label	<u>*PC ‹ label, if ACC.0 = 1; else PC = PC++*</u> If bit 0 of ACC is "1", The PC is replaced with "label", and a jump occurs. If bit 0 of ACC is "0", the PC is incremented. <u>Flag affected:</u> none	**JB0 EXIT** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> EXIT 1A9h 1A9h ACC xxx1b xxx1b PC --- 1A9h
JB1 label	<u>*PC ‹ label, if ACC.1 = 1; else PC = PC++*</u> If bit 1 of ACC is "1", The PC is replaced with "label", and a jump occurs. If bit 1 of ACC is "0", the PC is incremented. <u>Flag affected:</u> none	**JB1 EXIT** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> EXIT 1A9h 1A9h ACC xx0xb xx0xb PC 100h 101h
JB2 label	<u>*PC ‹ label, if ACC.2 = 1; else PC = PC++*</u> If bit 2 of ACC is "1", The PC is replaced with "label", and a jump occurs. If bit 2 of ACC is "0", the PC is incremented. <u>Flag affected:</u> none	**JB2 EXIT1** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> EXIT1 1B9h 1B9h ACC x1xxb x1xxb PC ---- 1B9h
JB3 label	<u>*PC ‹ label, if ACC.3 = 1; else PC = PC++*</u> If bit 3 of ACC is "1", The PC is replaced with "label", and a jump occurs. If bit 3 of ACC is "0", the PC is incremented. <u>Flag affected:</u> none	**JB3 EXIT2** <u>Memory.</u> <u>Before exec.</u> <u>After exec.</u> EXIT2 1C9h 1C9h ACC 1xxxb 1xxxb PC ---- 1C9h

JZ label	*PC ‹ label, if ACC = 0 (PSR.0 = 1); else PC = PC++* If the ACC is zero. The PC is replaced with "label", and a jump occurs. If ACC is not zero, the PC is incremented. Flag affected: none	**JZ LOOP1** Memory. exec. / Before exec. / After LOOP1 255h 255h ACC 0000b 0000b PC ---- 255h
JNZ label	*PC ‹ label, if ACC != 0 (PSR.0 = 0); else PC = PC++* If the ACC is not zero. The PC is replaced with "label", and a jump occurs. If ACC is zero, the PC is incremented. Flag affected: none	**JNZ LOOP2** Memory. exec. / Before exec. / After LOOP2 300h 300h ACC 0000b 0000b PC 120h 121h
JC label	*PC ‹ label, if CF = 1 (PSR.1 = 1); else PC = PC++* If carry flag (CF) is "1". The PC is replaced with "label", and a jump occurs. If CF is zero, the PC is incremented. Flag affected: none	**JC LOOP2** Memory. exec. / Before exec. / After LOOP2 300h 300h PSR.1 1 1 PC ----- 300h
JNC label	*PC ‹ label, if CF = 0 (PSR.1 = 0); else PC = PC++* If carry flag (CF) is zero. The PC is replaced with "label", and a jump occurs. If CF is "1", the PC is incremented. Flag affected: none	**JNC LOOP2** Memory. exec. / Before exec. / After LOOP2 300h 300h PSR.1 1 1 PC 200h 201h
JBZ1 label	*PC ‹ label, if BZ1 = 1 (PSR.2 = 1); else PC = PC++* If channel 1 is busy (BZ1 flag is set to 1), the PC is replaced with "label" and a jump occurs.Else,the PC is incremented. Flag affected: none	**JBZ1 AGAIN** Memory. exec. / Before exec. / After AGAIN 2A0h 2A0h PSR.2 1 1 PC ----- 2A0h

JBZ2 label	*PC ‹ label, if BZ2 = 1 (PSR.3 = 1); else PC = PC++* If channel 2 is busy (BZ2 flag is set to 1), the PC is replaced with "label" and a jump occurs. Else, the PC is incremented. Flag affected: none	**JBZ2 AGAIN** Memory. Before exec. After exec. AGAIN 2A0h 2A0h PSR.3 0 0 PC 1BFh 1C0h
CJNE Rn, i, label	*ACC ‹ Rn - i; PC ‹ label, if ACC != 0; else PC = PC++* Compare the content of Rn with an immediate value i. If not equal, the PC is replaced with "label" and a jump occurs. Else, the PC is incremented. Flag affected: ZF, CF	**CJNE R10, 1010b, ERROR** Memory. Before exec. After exec. i 1010b 1010b R10 1011b 1011b ERROR 1A0h 1A0h ACC --- 0001b PC --- 1A0h
CJE Rn, i, label	*ACC ‹ Rn - i; PC ‹ label, if ACC = 0; else PC = PC++* Compare the content of Rn with an immediate value i. If equal, the PC is replaced with "label" and a jump occurs. Else, the PC is incremented. Flag affected: ZF, CF	**CJE R10, 1010b, ERROR** Memory. Before exec. After exec. i 1010b 1010b R10 1011b 1011b ERROR 1A0h 1A0h ACC --- 0001b PC 0B9h 0BAh
CJNE Rn, WR, label	*ACC ‹ Rn - WR; PC ‹ label, if ACC != 0; else PC = PC++* Compare the content of Rn with that of WR.If not equal, the PC is replaced with "label" and a jump occurs. Else, the PC is incremented. Flag affected: ZF, CF	**CJNE R10, WR0, ERROR** Memory. Before exec. After exec. WR0 1010b 1010b R10 1011b 1011b ERROR 1A0h 1A0h ACC --- 0001b PC --- 1A0h
CJE Rn, WR, label	*ACC ‹ Rn - WR; PC ‹ label, if ACC = 0; else PC = PC++* Compare the content of Rn with that of WR. If equal, the PC is replaced with "label" and a jump occurs. Else, the PC is incremented. Flag affected: ZF, CF	**CJE R10, WR0, ERROR** Memory. Before exec. After exec. WR0 1010b 1010b R10 1011b 1011b ERROR 1A0h 1A0h ACC --- 0001b PC 0B9h 0BAh

DJNZ Rn, label	*ACC, Rn ‹ Rn - 1; PC ‹ label, if ACC != 0; else PC = PC++* Decrement Rn by 1. If ACC is not equal to zero, the PC is replaced with "label" and a jump occurs. Else, the PC is incremented. Flag affected: ZF, CF	**DJNZ R6, start** Memory. Before exec. After exec. start 60h 60h R6 4 3 ACC --- 3 PC --- 60h
DJZ Rn, label	*ACC, Rn ‹ Rn - 1; PC ‹ label, if ACC = 0; else PC = PC++* Decrement Rn by 1. If ACC is zero, the PC is replaced with "label" and a jump occurs. Else, the PC is incremented. Flag affected: ZF, CF	**DJZ R6, start** Memory. Before exec. After exec. start 60h 60h R6 1 0 ACC --- 0 PC --- 60h

Subroutine

CALL label	*STACK ‹ PC + 1, then PC ‹ label* The next PC (return entry) is stored in the STACK and then the direct address "label" is loaded into PC. A subroutine is called. Flag affected: none	**CALL OUTPUT** Memory. Before exec. After exec. OUTPUT 160h 160h PC 40h 160h STACK --- 41h
RTN	*PC ‹ STACK* The PC is restored from the STACK. A return from a subroutine occurs. Flag affected: none	**RTN** Memory. Before exec. After exec. STACK 160h ---- PC ---- 160h
RTI	*PC ‹ STACK* The PC is restored from the STACK. A return from an interrupt service routine (ISR) occurs. Flag affected: none	**RTN** Memory. Before exec. After exec. STACK 160h ---- PC ---- 160h

Others

NOP	No operation. Flag affected: none	**NOP**

END	The program stops execution and the W562xxx enters power-down mode. Flag affected: none	**END**
EN INT	This instruction enables the interrupt source. Flag affected: none	**EN INT**
DIS INT	This instruction disables the interrupt source. Flag affected: none	**DIS INT**
PLAY CH1/CH2, h4+vocice+t4[*]	Use channel 1 or channel 2 to play voice segments. The voice segmemts can be the following types : 1. Speech 2. PCM Melody 3. Silence These segments can be combined arbitrarily. Flag affected: BZ1/ BZ2	**1. PLAY CH2, h4+Bird+t4** (Speech) **2. PLAY CH1, h4+Sonatina+t4** (PCM Melody) **3. PLAY CH1, h4+[1A0B]+t4** (Silence)
STOP CH1/CH2	Stop the operation of channel 1 or channel 2 immediately. Flag affected: BZ1 / BZ2	**STOP CH1**
CLR TF0 / TF1	Clear the Trigger Flag of port 0 (TF0) or that of port 1 (TF1). Flag affected: TF0 / TF1	**CLR TF0**

[*] See **Speech Synthesis** and **Error! Reference source not found.** for more details.

Reserved Words

There are several reserved words for the W562xxx that should not be used as a normal label names. The following tables lists them.

ALU	**ADD, ADDC, OR, XOR, AND, SUB, SUBC, DEC, INC, NOT, SETB, CLRB, RORC, ROLC, SHRC, SHLC**
Data Move	**LD, LDR, MV**
Branch	**JP, JB0, JB1, JB2, JB3, JZ, JNZ, JC, JNC, JBZ1, JBZ2, CJNE, CJE, DJNZ, DJZ**
Subroutine	**CALL, RTN, RTI**
Other Instructions	**NOP, END, EN INT, DIS INT, PLAY, STOP, CLR,**
Register	**All RAM-Mapped Registers.**
Keyword	**POI, TG, TIMER_INTERRUPT, MACRO, ENDM, DEFINE, TEST**
Directives	**VOL 0 - VOL 7, FREQ 0 ~ FREQ 3**

SOFTWARE APPLICATIONS

This chapter provides explanations of how to use the W56000, W56xxx and instruction set features along with assembly language coding examples. Besides, some demo programs are also provided to demonstrate the functions. Users can get the object files from the example program directory and download directly to ICE system to run the programs.

Trigger Debounce

Since all P0 and P1 triggers (total 8 pins) share the same TG interrupt vector, users have to further differentiate the real one source from others by programming. Also, trigger debounce should be considered in the TG section.
The following two assembly source code provide explanations of how to do these works by software instruction.

Example 0-1 : Use only P0 as input pins. (For your reference only)

```
; Function :non_retriggerable
W562S20                         ; Body declaration
Macro INITIAL                   ;macro start
      LD MODE,0000B             ;
      LD IOR1,0000B             ;P1 as input port
      LD PCR1,1111B             ;
      LD IER,0100B              ;Enable TG
      LD PER0,1111B             ;Enable all P0 individual interrupt
ENDM
POI:                            ; "POI", a keyword, Reset entry vector
     INITIAL
```

```
        END
TG:                                     ;"TG", a keyword, Port 0 and Port 1 trigger interrupt entry
     MV WR0, P0                         vector
     NOP
     NOP
     NOP
     NOP
     NOP
     NOP                                ; NOP for S/W debounce time
     XOR P0, WR0
     JNZ Bounce
     MV WR1,TF0
     MV ACC,WR1
     JB0  KEY1
     JB1  KEy2
     JB2  KEY3
     JB3  KEY4
Bounce:
     CLR  TF0
      RTI
KEY1:
    PLAY CH1, H4+Do+T4
    JP  Chkbusy
KEY2:
    PLAY CH1, H4+RE+T4
    JP  Chkbusy
KEY3:
    PLAY CH1, H4+MI+T4
     JP  Chkbusy
KEY4:
    PLAY CH1, H4+FA+T4
Chkbusy:
    JBZ1  Chkbusy                       ; Check synthesizer active or not
    JB  Bounce                          ;
```

```
; Function :retriggerable
W562S20                         ; Body declaration
Macro INITIAL                   ;macro start
        LD MODE,0000B           ;
        LD IOR1,0000B           ;P1 as input port
        LD PCR1,1111B           ;
        LD IER,0100B            ;Enable TG
        LD PER0,1111B           ;Enable all P0 individual interrupt
ENDM                            ;marco end
POI:                            ; "POI", a keyword, Reset entry vector
       INITIAL
        END
TG:                             ;"TG", a keyword, Port 0 and Port 1 trigger interrupt entry
     MV WR0, P0                 vector
     NOP
     NOP
     NOP
     NOP
     NOP
     NOP                        ; NOP for S/W debounce time
     XOR P0, WR0
     JNZ Bounce
     MV WR1,TF0
     MV ACC,WR1
     JB0  KEY1
     JB1  KEy2
     JB2  KEY3
     JB3  KEY4
Bounce:
     CLR  TF0
      RTI
KEY1:
    PLAY CH1, H4+Do+T4
    JP  ReleaseKey
KEY2:
    PLAY CH1, H4+RE+T4
    JP  ReleaseKey
KEY3:
    PLAY CH1, H4+MI+T4
     JP  ReleaseKey
KEY4:
    PLAY CH1, H4+FA+T4
ReleaseKey:
    MV WR0,P0                   ; Check key is released or not
    XOR R0,1111B                ;
    JNZ  ReleaseKey
    JB  Bounce
```

Normally, the trigger debounce and key scan routines can be written as modules for most programs to use repeatedly. Only slight modifications are necessary to meet certain specific requirements**.**

Keypad Matrix Application

For applications that require a large keypad of many keys, the W562xxx family offers a direct row and column matrix of up to 8 x 9 keys.
This program uses port 0 and port 1 as input ports, while port 2 and port 3 as the output ports to form the keypad matrix of 8 x 8 keys. We can also add the Vss line as another column to get a maximum of 8 x 9 keypad matrix in a direct manner.
You might be able to further expand the keypad matrix by adding external resistors and diodes. This part will be collected in the application note later on.

Application Circuit

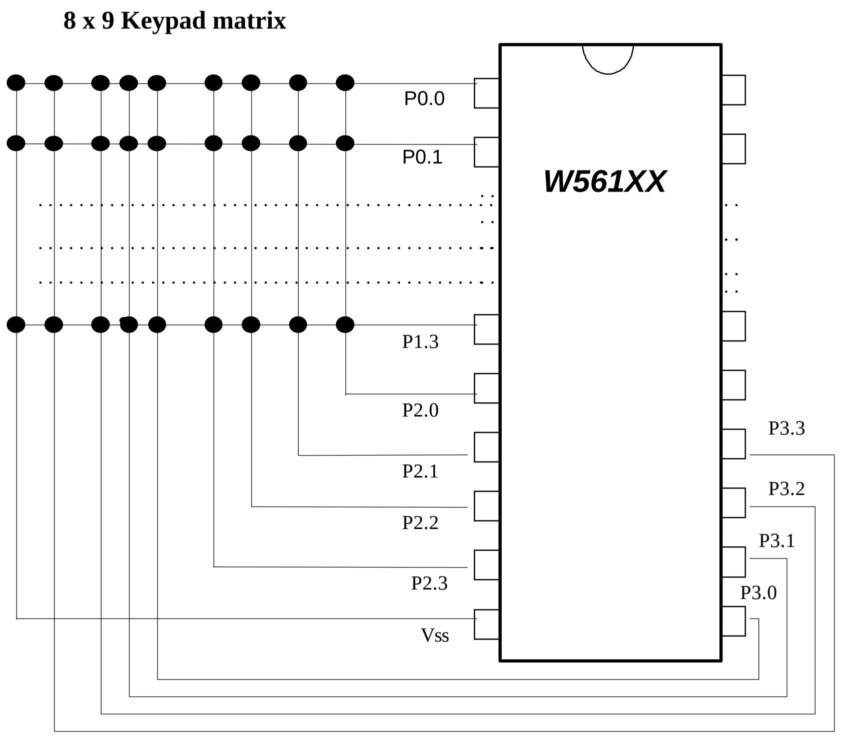


Figure 0-1. An 8 x 9 keypad matrix diagram

Version	Date	Writer	Reasons for change
A2	Oct. 14th, 1999	Sophia Ho	• Add two body with two chips solution W562M-04 W56000+W55M08 W562M-05 W56000+W55M06
A3	Nov. 18th, 1999	Sophia Ho	• modify TXF register table on page15 • remove the software application about volume control