Table of Contents

List of Figures........1
List of Tables........1
Acknowledgement........2
Section 1: Design Introduction........4
Section 2: Detailed Design Description........4
Section 2.1: LCD........4
Section 2.1.1: LCD Hardware Description........4
Section 2.1.2: LCD Software Description........6
Section 2.2: 4X4 Keypad........6
Section 2.2.1: Keypad Hardware Description........6
Section 2.2.2: Keypad Software Description........7
Section 2.3: FSM........11
Section 2.4: ATmega128........16
Section 2.4.1: ATmega128 clock frequency........17
Section 2.4.2: Timer 1 Hardware Description........17
Section 2.4.3: Waveform Generation Modes........22
Section 2.4.4: Fast PWM........22
Section 2.4.5: Timer1 Square-wave Software Description........26
Section 2.5: AD9834 Direct Digital Synthesis Device........28
Section 2.5.1: Direct Digital Synthesis........28
Section 2.5.2 AD9834 Hardware Description........32
Section 2.5.3: AD9834 Software Description........35
Section 2.6: MAX5402 Digital Potentiometer........39
Section 2.6.1: String Digital to Analog Converter (DAC)........39
Section 2.6.2: Digital Potentiometer........40
Section 2.6.3: MAX5402 Hardware Description........41
Section 2.6.4: MAX5402 Software Description........42
Section 2.7: LM6171 Op-Amp........43
Section 2.8: Serial Peripheral Interface (SPI)........44
Section 3: System Characteristics........45
Section 3.1: Sine characteristics........46
Section 3.2: Triangle characteristics:........46
Section 3.3: Square waveform:........46

Section 4: Troubleshooting 46
Section 4.1: Operating Checklist 46
Section 4.2: Cleaning..46
Section 4.3: Electrostatic Discharge (ESD) Precautions 47
Section 4.4: Troubleshooting Hints 47
Reference 49
Appendix 50
A1: Description of Bits in AD9834 Control Register 50
A2: Software 51

List of Figures

Figure 1: Function Generator Block Diagram 4
Figure 2: LCD view (Source: ESE381 Lecture#5) 5
Figure 3: DOG module (Source: ESE381 Lecture#5) 5
Figure 4: LCD module's connections to ATmega128 (Source: ESE381 Module#1 Lab manual) 6
Figure 5: 4X4 keypad interfacing ATmega128 7
Figure 6: Flow chart for Interrupt service routine (Source: AVR240 Application note) 9
Figure 7: State diagrams for generating sine or triangular waveform at different frequencies 13
Figure 8: State diagrams for generating square waveform at different frequencies and duty cycles 13
Figure 9: State diagram for adjusting the amplitude of sine or triangular waveform 14
Figure 10: Transitions from any state to sinOrTri_disp_Hz state and from any state to PWM_disp_freq 14
Figure 11: Transitions to sinOrTri_disp_Amplitude state 14
Figure 12: Schematic of ATmega128 Connections 17
Figure 13: Timer1 Block Diagram (Source: ATmega128 datasheet) 18
Figure 14: Prescaler for Timer1 (Source: ATmega128 datasheet) 19
Figure 15: Counter Unit Block Diagram (Source: ATmega128 datasheet) 19
Figure 16: Block Diagram of Output Compare Unit (Source: ATmega128 datasheet) 20
Figure 17: Schematic of Compare Match Output Unit (Source: ATmeaga128 datasheet) 21
Figure 18: Fast PWM Mode Timing Diagram (Source: ATmeaga128 datasheet) 23
Figure 19: Description of Control Bits (Source: ATmega128 datasheet) 26
Figure 20: AD9834 Functional Block Diagram (Source: AD9834 datasheet) 27
Figure 21: A Flexible DDS System (Source: Fundamental of DDS MT-085) 28
Figure 22: Digital Phase Wheel (Source: Fundamental of DDS MT-085) 29
Figure 23: Output Spectrum (Source: Fundamental of DDS MT-085) 30
Figure 24: Timing Characteristics of AD9834 (Source: AD9834 datasheet) 31
Figure 25: Initialization (Source: AD9834 datasheet) 35
Figure 26: Data Write (Source: AD9834 datasheet) 35
Figure 27: Flowchart for Initialization and Operation (Source: AD9834 datasheet) 36
Figure 28: MAX5402 Functional Diagram (Source: MAX5402 datasheet) 37
Figure 29: String DAC (Source: Basic DAC Architectures by Walt Kester) 38
Figure 30: Digital Potentiometer (Source: Basic DAC Architectures by Walt Kester) 39
Figure 31: Serial Interface Timing Diagram (Source: MAX5402 datasheet) 40
Figure 32: Adjustable Gain Amplifier (Source: Lab Module #5) 41
Figure 33: SPI interface between ATmega128 and Slave Units 42
Figure 34: LCD Block Diagram (Source: Lab Module #2) 45
Figure 35: Schematic for AD9834 wiring (Source: Lab Module #4) 46
Figure 36: Adjustable Gain Amplifier (Source: Lab Module #5) 46

List of Tables

Table 1: Logic Level Compatibility between ATmega128 and Keypad 7

Table 2: Clock Select Bit Description (Source: ATmega128 datasheet)....19
Table 3: Waveform Generation Mode Bit Description (Source: ATmeaga128 datasheet)....21
Table 4: Frequency Range Analysis....23
Table 5: Compare Output Mode, Fast PWM....24
Table 6: Frequency Range Analysis for AD9834....29
Table 7: Frequency Register Bits (Source: AD9834 datasheet)....33
Table 8: Writing to Frequency Registers (Source: AD9834 datasheet)....33

Acknowledgement

This work was supported by Electrical and Computer Engineering Department in Stony Brook University. The authors gratefully acknowledge Professor Kenneth Short and laboratory instructional specialist Scott Tierno.

Kenneth Short
Department of Electrical & Computer Engineering
State University of New York

Stony Brook, NY 11794-2350
e-mail:klshort@ece.sunysb.edu
Phone: (631) 632-8403
Fax: (631) 632-8494

Scott Tierno
Department of Electrical & Computer Engineering
Stony Brook University
Stony Brook, NY 11794-2350
e-mail: fstierno@stonybrook.edu

Section 1: Design Introduction

In this design, the overall system is composed of a ATmega128 microcontroller, a 3-line, 16 characters per line LCD display, a 4X4 keypad, a MAX5402 potentiometer, a AD9834 DDS and a LM6171 operational amplifier. To interface with users, LCD display and keypad are used. Timer/Counter 1 in ATmega128 is used to generate square wave in fast pulse width modulation mode. AD9834 DDS is used to generate sine or triangle waveform signal at different frequencies. MAX5402 and LM6171 will adjust the amplitude of the sine or triangle waveform signal. Detailed design description is provided in the following section and the complete software is provided in the Appendix. The following block diagram shows how each component in the system interface with each other.

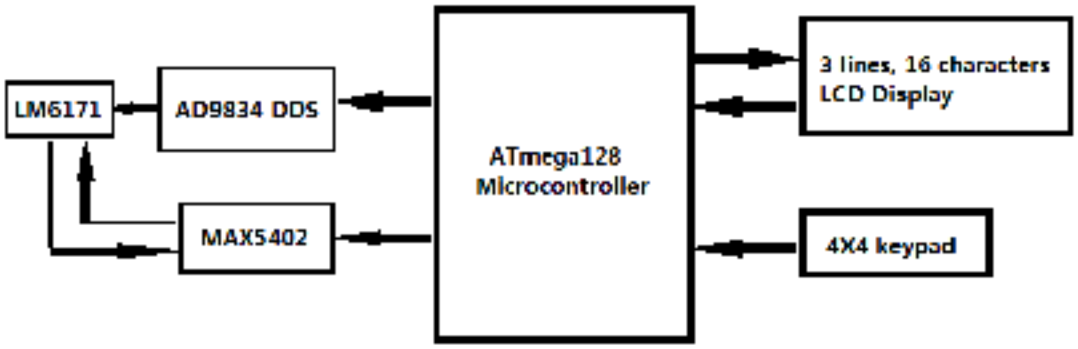


Figure 1: Function Generator Block Diagram

Section 2: Detailed Design Description

Section 2.1: LCD

Section 2.1.1: LCD Hardware Description

Liquid Crystal Displays (LCD) is an output device that allows user to view the setting of the system. In this project, an electronic assembly DOGM163W-A will be used to interface with

ATmega128 through the Sitronix ST7036 Dot Matrix Controller Driver. The LCD module displays 3 lines with 16 characters on each line. The backlight is white and the characters are black.

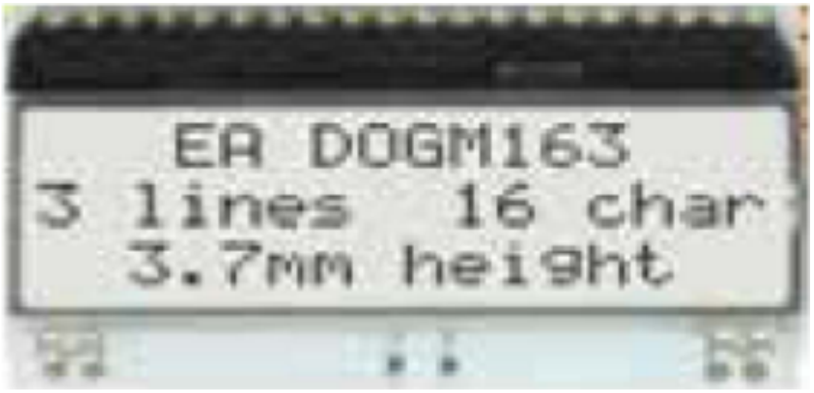


Figure 2: LCD view (Source: ESE381 Lecture#5)

Commands and data are sent to DOG module from ATmega128 through SPI interface. The ST7036 Dot Matrix Controller Drive will then controls what is displayed on the LCD. Figure 3 shows the interconnection between DOGM163W-A and ST7036 in DOG module. Figure 4 shows the interconnection between DOG module and ATmega128.

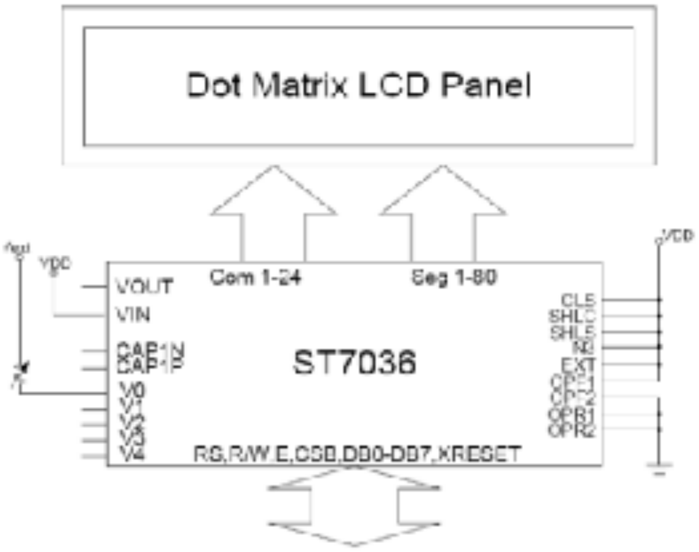


Figure 3: DOG module (Source: ESE381 Lecture#5)

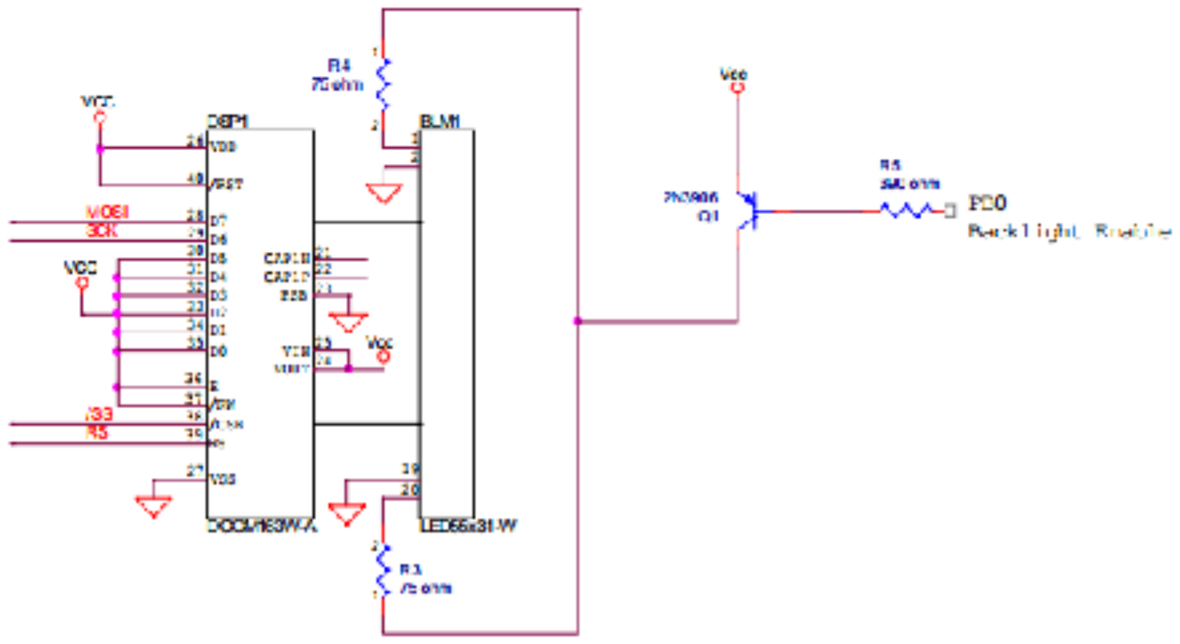


Figure 4: LCD module’s connections to ATmega128 (Source: ESE381 Module#1 Lab manual)

As shown in Figure 3, there are 4 pins that are connected to ATmega128. They are MOSI, SCK, /SS and RS. MOSI provides the serial data to the display and SCK will clock in the data. RS is for the register select signal. /SS will select the slave. These four data signals will be explained in details in SPI section.

Section 2.1.2: LCD Software Description

In this project, LCD module communicates with ATmega128 through write-only SPI interface. The asm file “lcd_dog_asm_driver_128” provides the functions that initialize and update DOG text. There are two functions that could be accessed from outside the file: init_lcd_dog and update_lcd_dog.
Init_lcd_dog will initialize DOG module LCD display for SPI operation. Update_lcd_dog will update the LCD display lines 1, 2, and 3, using the contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.
dsp_buff_1, dsp_buff_2 and dsp_buff_3 are the 3 arrays that store the 16 characters for each line. They are declared as public so programmers could write the characters by putting the character in the desired position of an array.

The LCD library is provided in the laboratory module. As such, detailed description of the library is not presented here. The library is written in Assembly language and the list file of the library is provided in the appendix section.

Section 2.2: 4X4 Keypad

Section 2.2.1: Keypad Hardware Description

Keypad is an input device that allows user to enter the command and data to ATmega128. In this project, this keypad is used to configure the output waveform. When user presses a key in the keypad, ATMega128 will wake up from power-down mode. The following block diagram describes the interconnection between keypad and ATmega128.

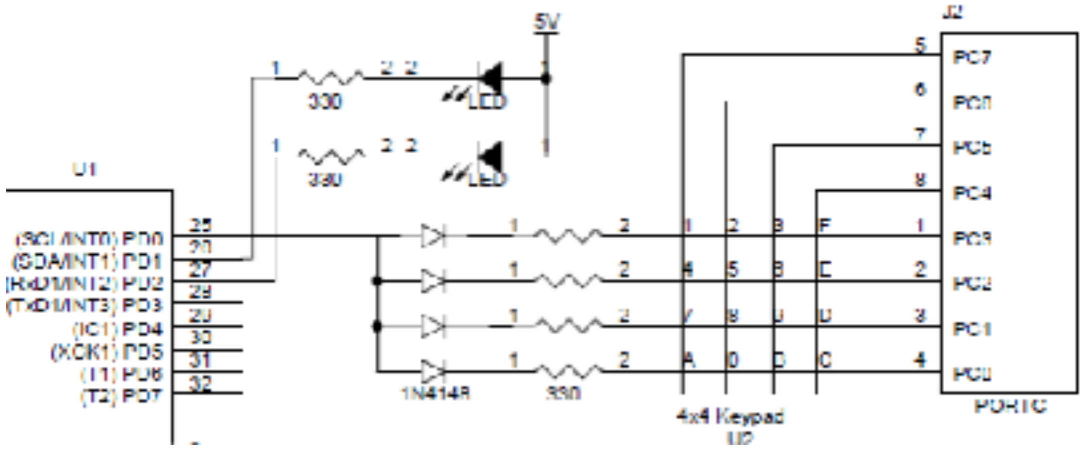


Figure 5: 4X4 keypad interfacing ATmega128

The four rows of keypad are connected to the low nibble of PINB from ATmega128. The four columns of keypad are connected to the high nibble. All the pins for PORTC and INT0 have their internal pull-up on. When a key is pressed, one of the diodes and one of the resistors will pull down INT0, which will cause interrupt in ATmega128. To make sure the logic level at INT0 is compatible with ATmega128 port pin electrical characteristics, the voltage at INT0 is measured when no key is pressed and any key is pressed. Table 1 below verifies the compatibility between keypad and ATmega128.

Table 1: Logic Level Compatibility between ATmega128 and Keypad

INT0	ATmega128 Port pin electrical characteristics	$V_{OH} > V_{IHmin}$ or $V_{OL} < V_{ILmax}$
V_{OH}: 5.312V	V_{IHmin}: 0.6Vcc=3V	Yes
V_{OL}: 0.6563V	V_{ILmax}: 0.2Vcc=1V	Yes

Section 2.2.2: Keypad Software Description

To interface the keypad with ATmega128, PORTC and INT0 (PD0) will be used. In the main program, INT0 and the low nibble of PORTC will be configured as input and their pull-up will be turned on. The high nibble of PORTC will be configured as outputs. Then the microcontroller will be placed in power-down mode and the system will enable the global interrupt and interrupt INT0. The C codes are shown below:

```
// Configure PortD for INT0 interrupt.
DDRD = 0xFE;                 // INT0 input
PORTD = 0x01;              // INT0 pullup enabled

  // Configure PortC for keypad, initial configuration
 DDRC = 0xF0;        // High nibble outputs, low nibble inputs
PORTC = 0x0F;
```

```
// Configure PortB for DOG LCD module's SPI interface
DDRB = 0xFF;            // Set PortB to outputs
SETBIT (PORTB, 0);      // unassert slave select
MCUCR = 0x30;    // Sleep enabled for power down mode.
EIMSK = 0x01;    // Enable interrupt INT0.
```

When the user press a key on the keypad, the interrupt will be driven low and the system will call interrupt service subroutine for INT0. In the interrupt service routine, key matrix is scanned and the key is encoded using a table lookup. The flowchart in Figure 2 shows the scanning and encoding process. When the encoded key is found, the program will use the encoded key as an index in a lookup table to find what the key is. The table is shown below:

```
const char tbl[16] = {'1', '2', '3', '$', '4', '5', '6', '*', '7', '8', '9',  ']', '%', '0', '[', '&'};

// '$' represents up arrow
// '*' represents down arrow
// '%" represents clear
// '&' reprensents enter
//'[' represents PWM/sinTri
//']'represents amplitude
```

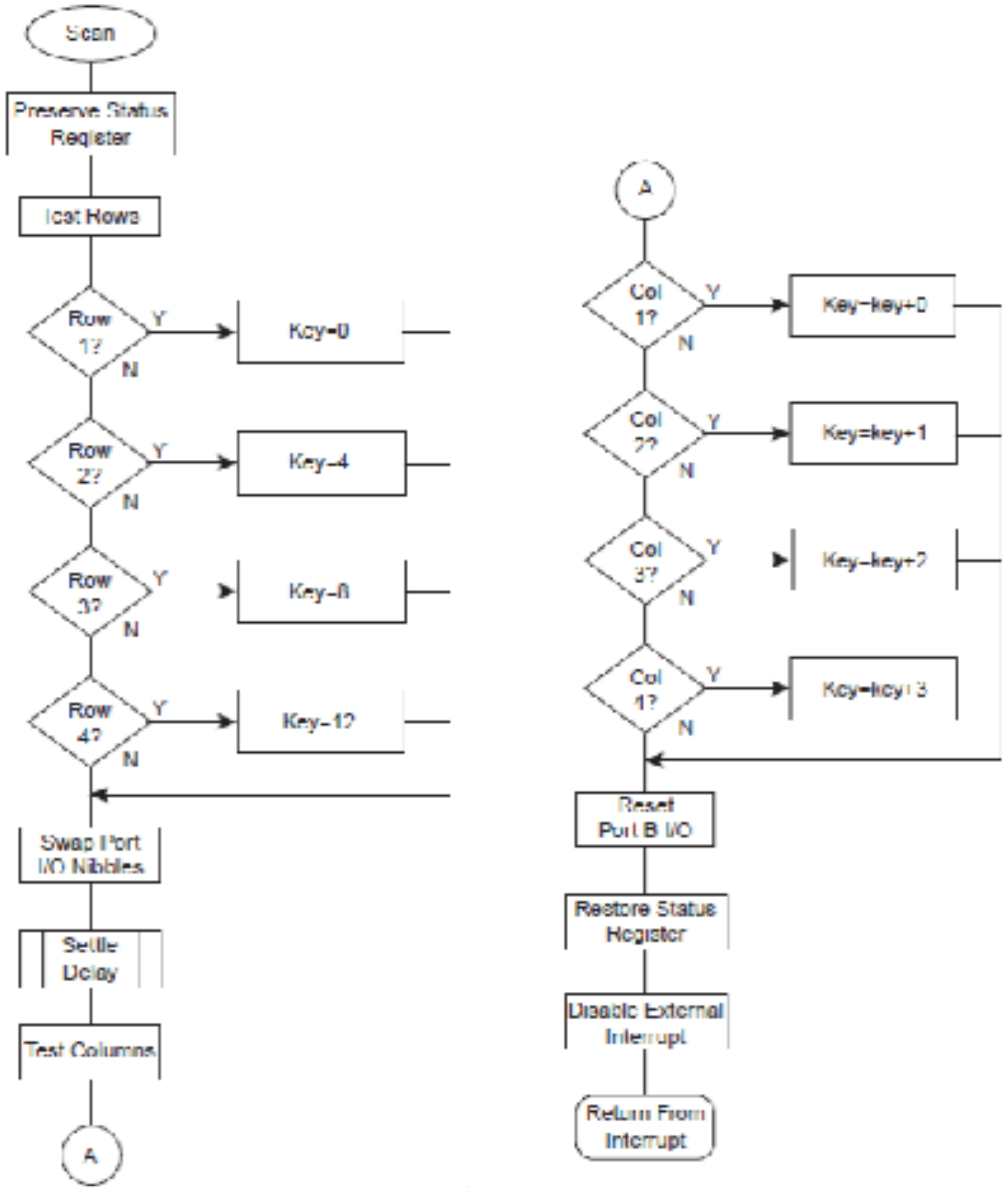


Figure 6: Flow chart for Interrupt service routine (Source: AVR240 Application note)

When a key is pressed, the system will execute the interrupt service routine and using the flow chart shown above to find the corresponding pressed key. The program first finds out the row of the pressed key and then reconfigures PORTC to find out the column of the pressed key.

Section 2.2.2.1: Keypad Parsing

The parsing of the keypad entries are done by saving the string value in the display buffer and then converting it into integer value. This is done by using a nesting loop that converts each character in the string into integer digit in the outer loop. Then each digit is multiplied by 10^P where P is the position of each of the digit starting from the least significant digit. This is done in the inner loop. The value of P starts at 0 for the least significant digit. The keypad parsing function is shown below.

```
// This function will convert the frequency value for sine or triangular
waveform
// on lcd to an integer
//*****************************************************************************
***

int getSineTriangularFrequency(void)
{

    //set frequency to 0 initially
    //outside the method,we can check if frequency is invalid or not by
checking
    //if frequency is equal to 0 or not;
    sineTriangularFrequency=0;
    int indexCopy=index; //save a copy of index
    int temp=0;

    if(index==9)
      ;

    //as long as i doesn't exceed the number of digits,loop continues
    for(int i=0;i<index-9;i++)
    {
        indexCopy--; //move indexCopy to the previous digit of number
        temp=dsp_buff_1[indexCopy]-'0';  //find the integer value of the bit

       //the inner for loop will multiply the bit by 10 depends on the
position
      for(int j=0;j<i;j++)
      {
        temp*=10;
      }
      //add the value of that bit
      sineTriangularFrequency+=temp;
    }


      int x=0;
      return sineTriangularFrequency;
}
```

Section 2.2.2.2: Updating Display Buffer

The function int putchar(int c) is used to update the display buffers when a number is entered or deleted. The precondition for the function is that the passing parameter could only be a number or a character representation of the “delete” key.
The pointer index helps determine the position of the new character in a line buffer. As described in LCD section, there are 3 line buffers for display: dsp_buff_1, dsp_buff_2 and dsp_buff_3. When index is smaller than 16, dsp_buff_1 is updated. When index is bigger than 16 and smaller than 32, dsp_buff_2 is updated. Otherwise, dsp_buff_3 is updated.
When the passing value doesn’t represent “delete”, the key value is saved in the line buffer. The index is then incremented.
When the passing value is the character representation of “delete”, index is decremented first and then the content in the position is cleared.

```
// This function displays a single ascii chararacter c on the lcd at the
// position specifieb by the global variable index
// NOTE: update_dsp must be called after to see results
//
//
// Modified
//***************************************************************************
***

int putchar(int c)

{

  if (index < 16)
    {
      if(c!='%')
        dsp_buff_1[index++] = (char)c;
      else
       {
         if(index<=9)
           ;
         else
         dsp_buff_1[--index] = 0;}
     }
  else if (index < 32)
    {
      if(c!='%')
        dsp_buff_2[index++ - 16] = (char)c;
      else
       {
         dsp_buff_2[--index-16] = 0;}
        }
  //If third line is somehow reached, the system will do nothing
  //The only way to resume is to press up arrow or down arrow
  else
  {
    if(c!='%')
    dsp_buff_3[index++] = (char)c;
    else
      dsp_buff_3[--index-32] = 0;;
  }



  return c;
}
```

Section 2.3: FSM

In our system, the user interface is implemented in table driven finite state machine. There are total 10 states, which are shown below. Four states are for setting the frequency of sine or triangular waveform, four states are for generating square wave at different frequencies and duty cycles, and last two states are for adjusting the output amplitude of sine or triangular wave.

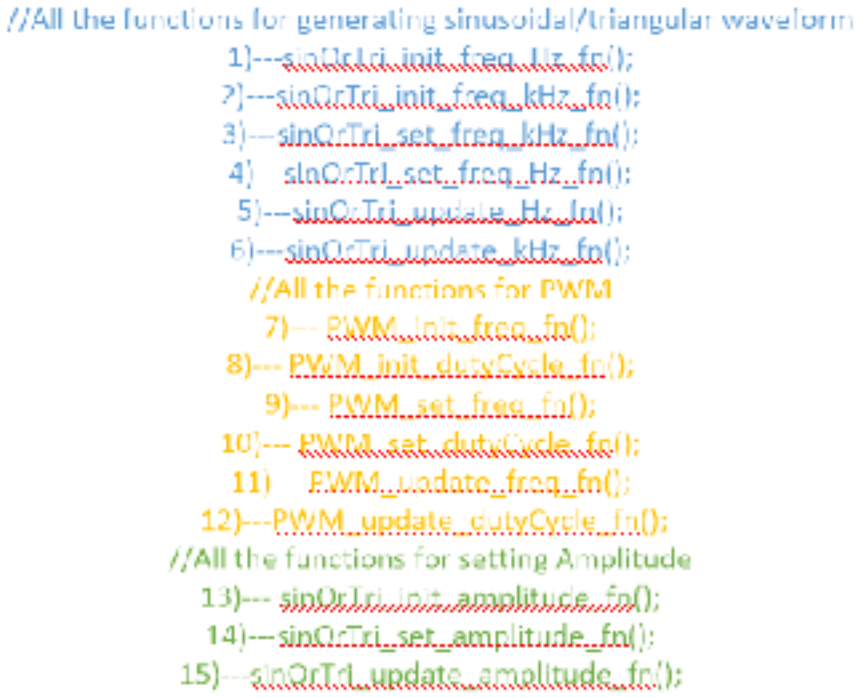
Generating sine or triangular waveform at different frequency

```
sinOrTri_disp_Hz     //initialize the lcd display to ask user to enter frequency value in Hz
sinOrTri_disp_kHz    //initialize the lcd display to ask user to enter frequency value in kHz
sinOrTri_set_kHz     //adjust and set the frequency value in kHz
sinOrTri_set_Hz      //adjust and set the frequency value in Hz
```

Generating square wave at different frequencies and

PWM_disp_freq   //initialize the lcd display to ask user to enter frequency value in Hz
PWM_disp_dutyCycle //initialize the lcd display to ask user to enter duty cycle
 PWM_set_dutyCycle //adjust and set the duty cycle
PWM_set_freq   //adjust and set the frequency
Adjusting amplitude for sine or triangular wave
 sinOrTri_disp_Amplitude   //initialize the lcd display to ask user to enter amplitude value
sinOrTri_set_Amplitude.   //adjust and set the amplitude

Since there are 10 states and many state transitions, the states diagram is divided into 3 categories as shown above. Also, to make the state diagrams concise, the function tasks are represented by numbers.



```
//All the functions for generating sinusoidal/triangular waveform
1)---sinOrTri_init_freq_Hz_fn();
2)---sinOrTri_init_freq_kHz_fn();
3)---sinOrTri_set_freq_kHz_fn();
4)   sinOrTri_set_freq_Hz_fn();
5)---sinOrTri_update_Hz_fn();
6)---sinOrTri_update_kHz_fn();
//All the functions for PWM
7)--- PWM_init_freq_fn();
8)--- PWM_init_dutyCycle_fn();
9)--- PWM_set_freq_fn();
10)--- PWM_set_dutyCycle_fn();
11)    PWM_update_freq_fn();
12)---PWM_update_dutyCycle_fn();
//All the functions for setting Amplitude
13)--- sinOrTri_init_amplitude_fn();
14)---sinOrTri_set_amplitude_fn();
15)---sinOrTri_update_amplitude_fn();
```

The state diagrams for this finite state machine are shown below.

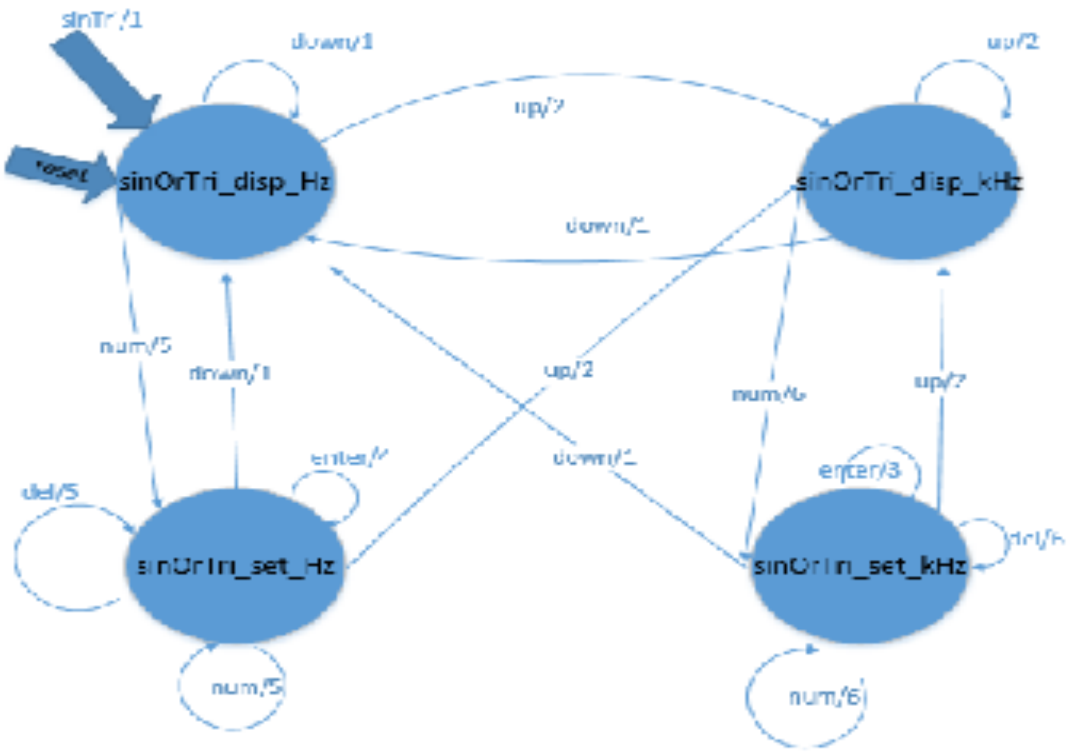


Figure 7: State diagrams for generating sine or triangular waveform at different frequencies

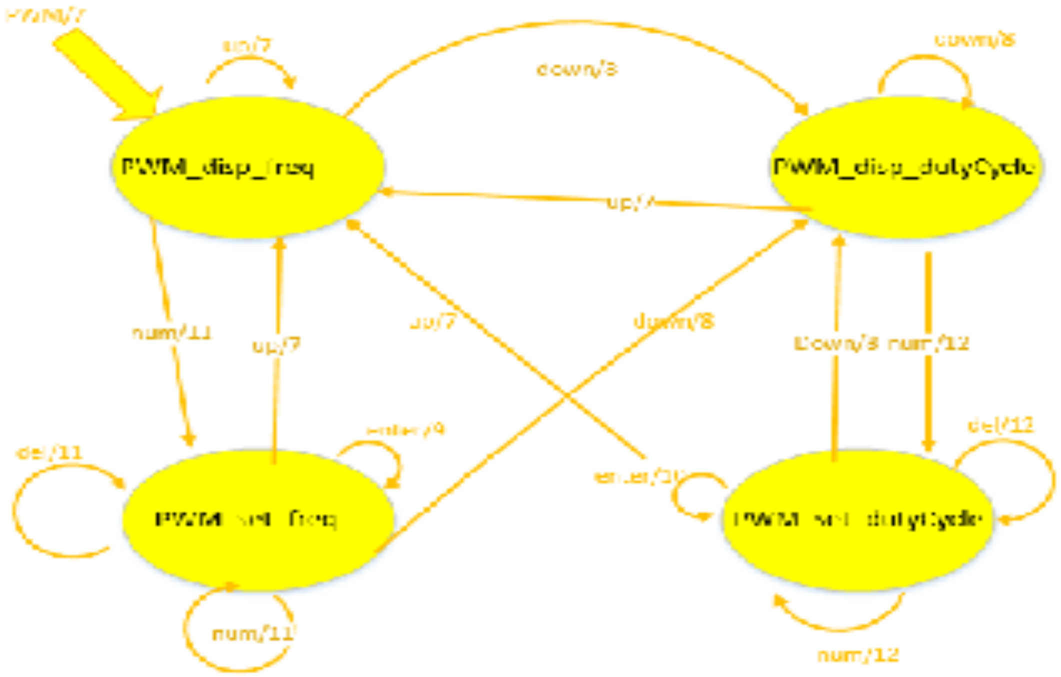


Figure 8: State diagrams for generating square waveform at different frequencies and duty cycles

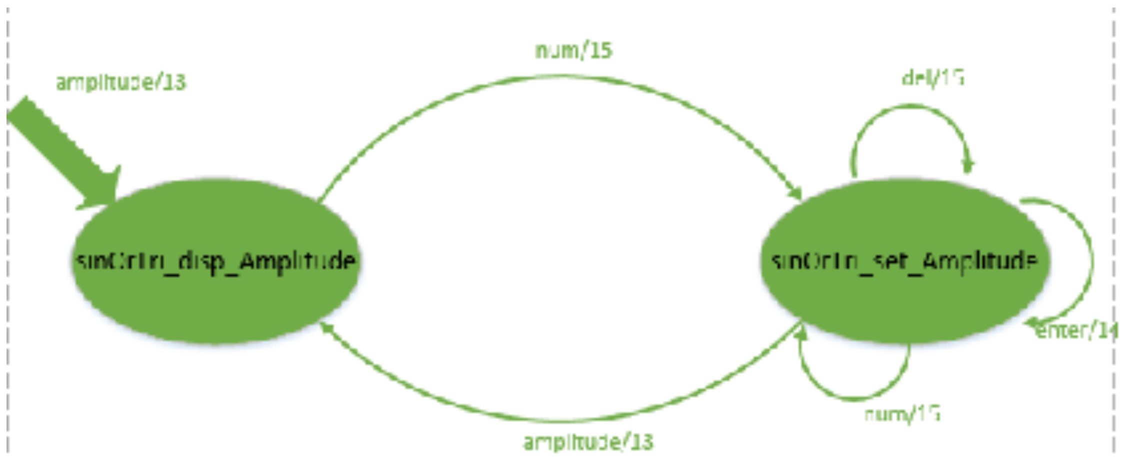


Figure 9: State diagram for adjusting the amplitude of sine or triangular waveform

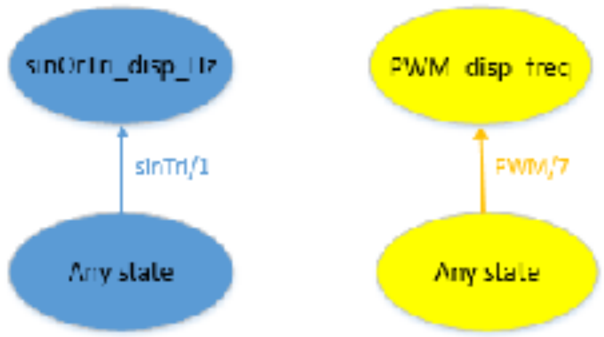


Figure 10: Transitions from any state to sinOrTri_disp_Hz state and from any state to PWM_disp_freq

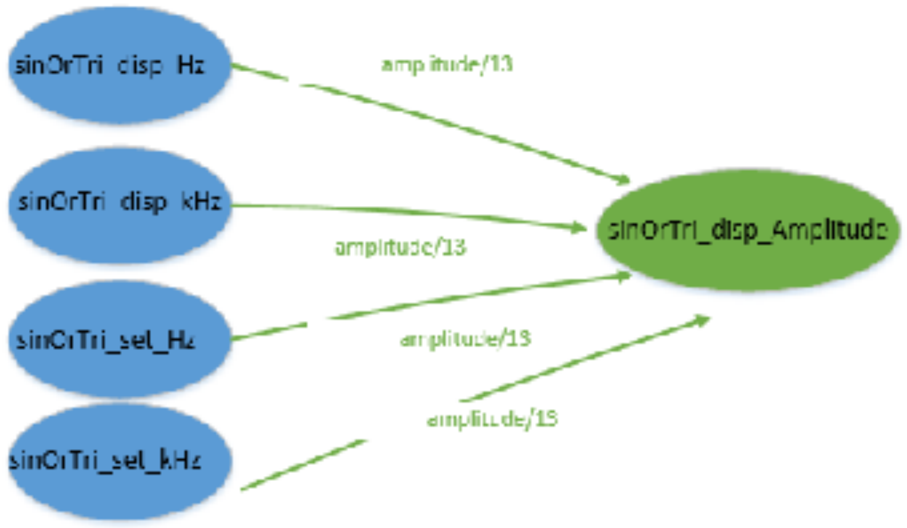


Figure 11: Transitions to sinOrTri_disp_Amplitude state

As stated previously, the finite state machine is implemented in a table driven fashion. In order to implement the table, the states and input keys are declared first by using enum type.

```
// states in the FSM
typedef enum{sinOrTri_disp_Hz, sinOrTri_disp_kHz,  sinOrTri_set_kHz, sinOrTri_set_Hz,
PWM_disp_freq, PWM_disp_dutyCycle,PWM_set_dutyCycle, PWM_set_freq,
sinOrTri_disp_Amplitude,sinOrTri_set_Amplitude}  state ;


// keys on the keypad, eol is a psuedo key used as a default in the state table
typedef enum {num, del, enter, up, down, PWM, sinTri, amplitude, eol} key ;
```

Then all the functions in this finite state machine are declared. Note that all functions have the same signatures and return type.

```
//All the functions for generating sinusoidal/triangular waveform
extern void sinOrTri_init_freq_Hz_fn();
extern void sinOrTri_init_freq_kHz_fn();
extern void sinOrTri_set_freq_kHz_fn();
extern void sinOrTri_set_freq_Hz_fn();
extern void sinOrTri_update_Hz_fn();
extern void sinOrTri_update_kHz_fn();
//All the functions for PWM
extern void PWM_init_freq_fn();
extern void PWM_init_dutyCycle_fn();
extern void PWM_set_freq_fn();
extern void PWM_set_dutyCycle_fn();
extern void PWM_update_freq_fn();
extern void PWM_update_dutyCycle_fn();
//All the functions for setting Amplitude
extern void sinOrTri_init_amplitude_fn();
extern void sinOrTri_set_amplitude_fn();
extern void sinOrTri_update_amplitude_fn();
```

In order to use the task function, the pointer to a function task is declared.

```
typedef void (* task_fn_ptr) ();
```

Next, a structure transition is declared to represent one row of a state transition. The transition is consisted of an input key value, the next state, and a function task pointer.

```
typedef struct {
 key keyval;
state next_state;
task_fn_ptr tf_ptr;
} transition;
```

Next, all the possible transitions for all states are declared. For example, the transitions for sinOrTri_disp_Hz are shown below. Structure transition type is used for the transitions table.

```
// subtable for sinOrTri_disp_Hz state
```

```
const transition sinOrTri_disp_Hz_transitions [] =
{
// INPUT      NEXT_STATE       TASK
   {num,      sinOrTri_set_Hz,  sinOrTri_update_Hz_fn},
   {down,     sinOrTri_disp_Hz, sinOrTri_init_freq_Hz_fn},
  {up,       sinOrTri_disp_kHz,sinOrTri_init_freq_kHz_fn},
 {PWM,      PWM_disp_freq,    PWM_init_freq_fn},
 {sinTri,    sinOrTri_disp_Hz, sinOrTri_init_freq_Hz_fn},
 {amplitude, sinOrTri_disp_Amplitude,sinOrTri_init_amplitude_fn},
{eol,       sinOrTri_disp_Hz, null_fn}
};
```

Finally, the pointer to one of the structure transition is declared.

```
const transition * ps_transitions_ptr[10] =
{sinOrTri_disp_Hz_transitions,
sinOrTri_disp_kHz_transitions,
sinOrTri_set_kHz_transitions,
sinOrTri_set_Hz_transitions,
PWM_disp_freq_transitions,
PWM_disp_dutyCycle_transitions,
PWM_set_dutyCycle_transitions,
PWM_set_freq_transitions,
sinOrTri_disp_Amplitude_transitions,
sinOrTri_set_Amplitude_transitions };
```

After all the declarations, a loop is used to search the array of transition structures corresponding to the present state for the transition structure that has keyvalue field value that is equal to the current input key value or eol. When the current state and current input key value is found in the table, the function associated with the state transition is executed. Finally, next state of the transition is assigned to current state.

```
for (; (ps_transitions_ptr[ps][i].keyval != keyval)&& (ps_transitions_ptr[ps][i].keyval !
= eol); i++);
ps_transitions_ptr[ps][i].tf_ptr();
present_state = ps_transitions_ptr[ps][i].next_state;
```

Section 2.4: ATmega128

At the heart of the function generator is the microcontroller ATmega128. It is used to perform all the necessary calculations and decisions that combine to create the core of the overall system. The details of the ATmega128 architecture and capabilities are too extensive to cover in this document. As such only the utilized features of ATmega128 will be discussed here. The ATmega128 is mounted on a stamp board. The following schematic shows how the ATmega128 is connected to the board.

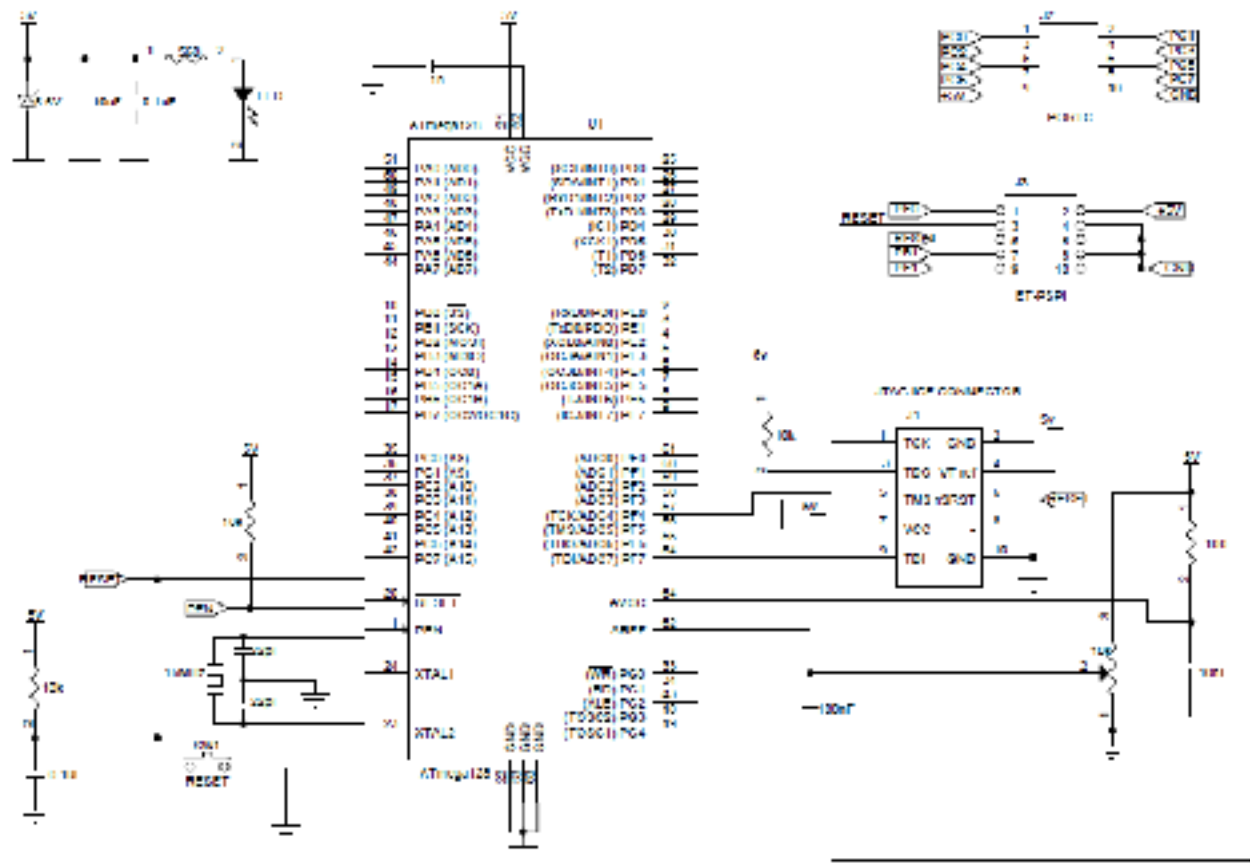


Figure 12: Schematic of ATmega128 Connections

Section 2.4.1: ATmega128 clock frequency

The clock frequency of ATmega128 is configured as 16MHz in this project. When the clock frequency of ATmega128 is configured in AVR studio to be 258CK+64ms using its external crystal, the time period for executing statement “sbi” is 128ns from the oscilloscope. Since these two instructions each takes two cycles, so one cycle will be 64ns. Therefore, the clock frequency of microcontroller will be about 15.625MHz (1/64ns), which is closed to 16MHz.

Section 2.4.2: Timer 1 Hardware Description

Timer/counter is a hardware counter that counts clock pulses from an internal or an external clock source. Timers operate independently of program execution. Timers are used to determine elapse time by counting the number of pulses during an interval and multiplying that number by the timer/counter’s clock period. Timers can be used to generate interrupts at certain intervals and perform time-sensitive tasks. In addition to measuring elapsed time the timer/counter can be used to generate periodic waveforms, hardware delays, pulse width modulation etc. The ATmega128 has four independent timer/counters. Timer 0 and 2 are 8-bit asynchronous and synchronous timer respectively. Timer 1 and 3 are 16-bit synchronous timers. In this design, the Timer 1 of the ATmega128 is used to generate the square wave of varying duty cycles. The block diagram of the timer 1 is shown below.

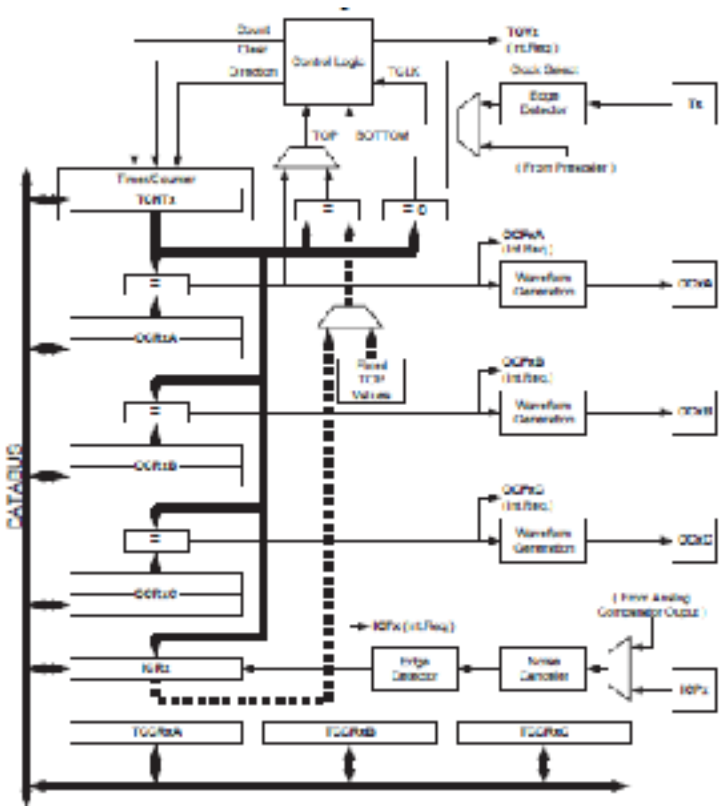


Figure 13: Timer1 Block Diagram (Source: ATmega128 datasheet)

The Timer 1 has three independent output compare units and double buffered output compare registers. When the counter's value matches the value in an output compare register (OCRn) the output compare flag (OCFn) in the Timer Interrupt Flag Register (TIFR) is set. The timer 1 also has one input capture unit, an input capture noise canceler, external event counter, frequency generator, and five independent interrupt sources including timer overflow (TOV1) and input capture interrupt (ICF1).

The Timer 1 also has Timer/Counter Register (TCNT1) which is 16-bit wide and gives direct access, both for read and write operations. This register holds the value of the ongoing count. The Output Compare Registers contains 16-bit values that are continuously compared with the counter value (TCNT1) and a match can be used to generate an output compare interrupt or to generate a waveform output on the OC1A/B/C pin. The compare match event will also set the compare match flag (OCF1A/B/C). The maximum Timer/Counter value can be defined by either the OCR1A register or the ICR1 register. This depends on the mode of operation and in this case the maximum value is set by ICR1 register in the Fast PWM mode and the OCR1A is used as the PWM output.

The Timer 1 can be clocked by an internal or an external clock source. The clock source can be selected the by manipulating the Clock Select (CS12:0) bits in the Timer/Counter Control Register b (TCCR1B). The Clock Select control bits can be modified according to the following table.

Table 2: Clock Select Bit Description (Source: ATmega128 datasheet)

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

The following diagram shows the prescaler for Timer/Counter 1, Timer/Counter 2, and Timer/Counter 3.

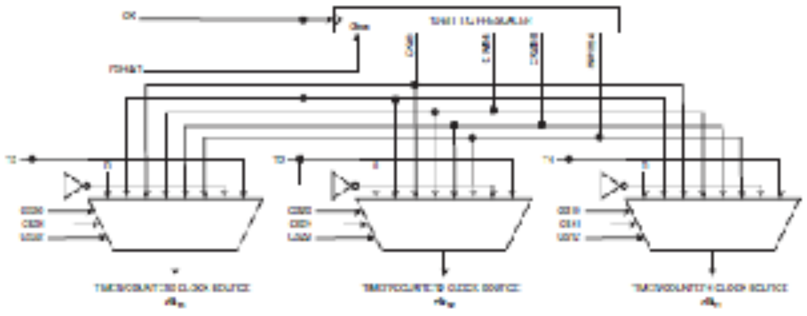

CK
PSR321
T1
T2
T3
TIMER/COUNTER1 CLOCK SOURCE
TIMER/COUNTER2 CLOCK SOURCE
TIMER/COUNTER3 CLOCK SOURCE

Figure 14: Prescaler for Timer1 (Source: ATmega128 datasheet)

Depending on the Clock Select bits the system clock can be divided by a prescaling factor. In the function generator only the system clock is used with no prescaling so only the CS10 bit is set. Since the Input Capture Unit and an external clock source are not used in the design their functions will not be discussed in this report.

At the heart of the Timer 1 is a 16-bit bi-directional counter unit. The block diagram of the counter unit is presented below.

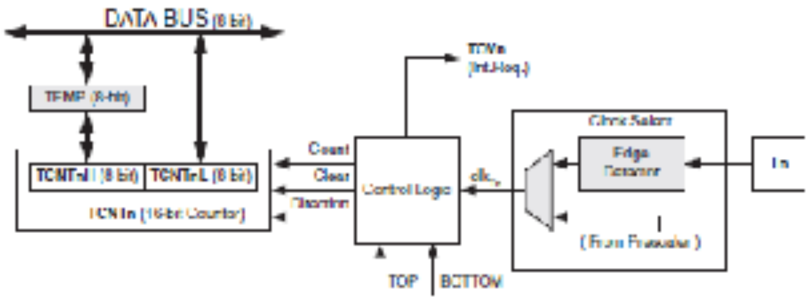


Figure 15: Counter Unit Block Diagram (Source: ATmega128 datasheet)

The TCNTnH register contains the upper 8-bits of the counter and the TCNTnL contains the lower 8-bits. The TCNTnH register can only be read by the CPU via the 8-bit TEMP register which is updated when the TCNTnL is read. The TEMP is also updated when TCNTnL is written and as such the CPU reads or writes the entire 16-bit within one clock cycle. The content of the TCNTn register is either incremented, decremented or cleared at each clock (clk_{Tn}) depending on the mode described previously. Tn is the external pin that connects to the external clock source.

The content of the TCNTn register is compared with the OCR1A/B/C registers using Output Compare Units. A block diagram of an Output Compare Unit is shown below.

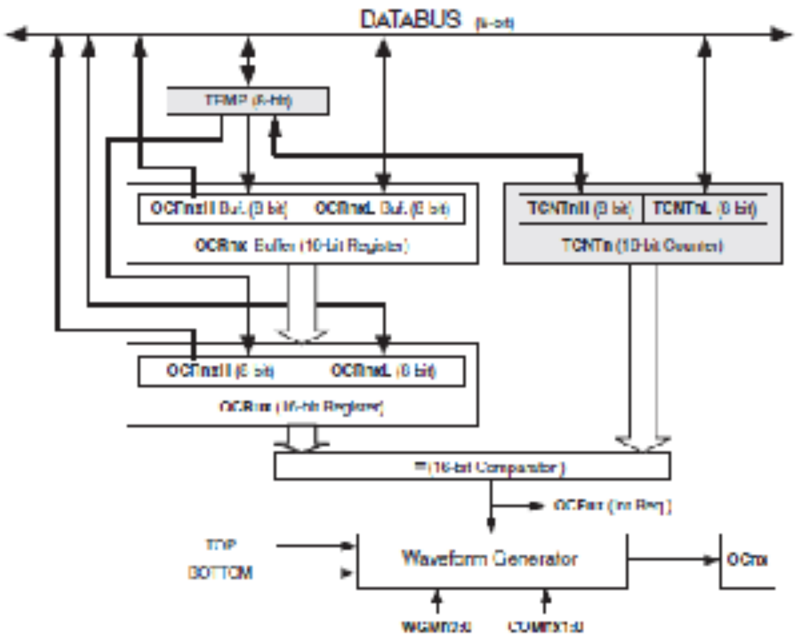


Figure 16: Block Diagram of Output Compare Unit (Source: ATmega128 datasheet)

A match will set the OCF1A/B/C flag in the next timer clock cycle and if the output compare match interrupt is enabled (OCIEnx=1) then flag generates an interrupt. The flag is automatically cleared when the interrupt is executed or writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate and output according to the operating mode set by the Waveform Generation mode and the Compare Output mode.

The Compare Output mode (COMnx1:0) bits are used by the waveform generator for defining the output compare state at the next compare match. The bits are also used to control the OCnx pin output source. A simplified schematic of the Compare Match Output Unit is shown below.

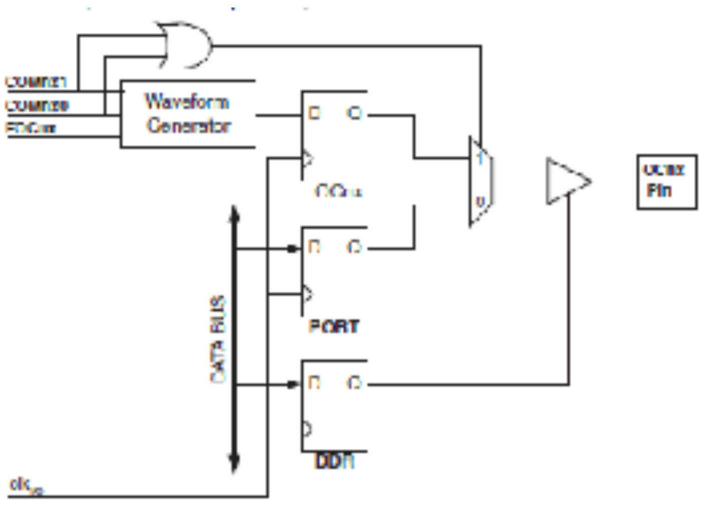


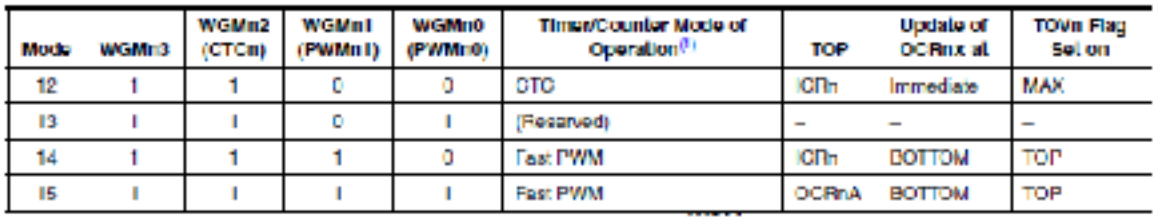
Figure 17: Schematic of Compare Match Output Unit (Source: ATmeaga128 datasheet)

If either of the COMnx1:0 bits are set the I/O port function is overridden by the output compare (OCnx) from the Waveform Generator. However, the direction of the OCnx pin is determined by the Data Direction Register (DDR) for the port pin.

The counting sequence is determined by the Waveform Generation mode bits (WGMn3:0) and the Compare Output mode bits (COMnx1:0) located in the TCCR1A and TCCR1B registers. The wave generation mode bits affect the counting sequence and the compare output mode bits control the PWM outputs. The modes bits can be set according to the following table.

Table 3: Waveform Generation Mode Bit Description (Source: ATmeaga128 datasheet)

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation[1]	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM



Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation[1]	TOP	Update of OCRnx at	TOVn Flag Set on
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

The function generator only utilizes the mode 14, Fast PWM, of the available wave generation modes. As such other modes will be discussed very briefly in this report.

Section 2.4.3: Waveform Generation Modes

In the Normal mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when in passes the maximum 16-bit value and restarts of 0x00. In normal operation the Timer/Counter overflow flag will be set. In the Clear Timer on Compare or CTC mode the OCR1A/B/C register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value matches the OCR1A/B/C. This mode allows greater control of the compare match output frequency. The Phase Correct PWM mode provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM (see the chart for exact values). In noninverting Compare Output mode, the output compare is cleared on the compare match between TCNT0 and OCR0 while counting up and set on the compare match while down counting. In inverting Output Compare mode, the operation is inverted. The Phase and Frequency Correct PWM mode operates just like the Frequency Correct PWM mode. The main difference between the phase correct and the phase and frequency correct PW mode is the time the OCRnx Register is updated by the OCRnx buffer Register.

Section 2.4.4: Fast PWM

The Fast PWM mode provides a mean of high frequency PWM waveform generation. It is a single-slope operation where the counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the output compare (OCnx) is cleared on the compare match between TCNTn and OCRnx, and set at BOTTOM. In inverting compare output mode output is set on compare match and cleared at BOTTOM. Since this is a single-slope operation the frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. Here the resolution for the fast PWM is set by OCR1A. The PWM resolution can be calculated using the following equation.

$$R_{FPWM} = \frac{\log(TOP+1)}{\log(2)}$$

In this mode the counter is incremented until the counter value matches the MAX value based on (WGM3:0) or the values in ICR1 or OCR1A. The counter is then cleared at the following timer clock cycle. A timing diagram for the fast PWM mode is shown below.

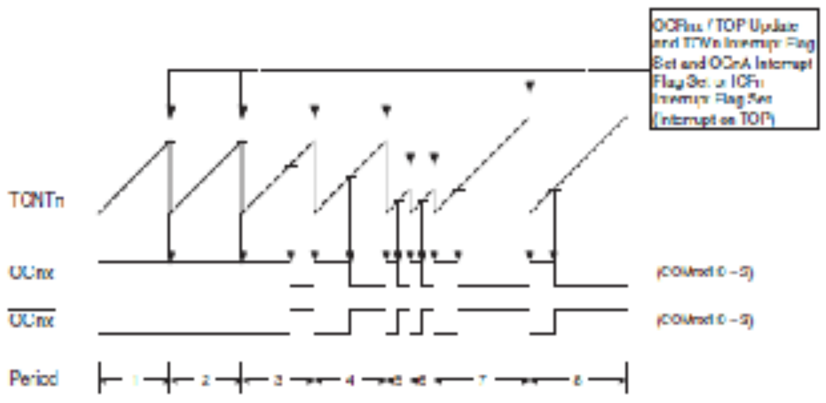

OCRnx / TOP Update and TOVn Interrupt Flag Set and OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)
TCNTn
OCnx
$\overline{\text{OCnx}}$
(COMnx1:0 = 2)
(COMnx1:0 = 3)
Period
1
2
3
4
5
6
7
8

Figure 18: Fast PWM Mode Timing Diagram (Source: ATmeaga128 datasheet)

In the figure the ICR1 is used to define the TOP and each time the counter reaches TOP the Timer/Counter Overflow Flag (TOV1) is set along with ICF1 flag. If one of the interrupts is enabled, the interrupt service routine can be used to update the TOP values. Care must be taken when updating the TOP value – if the TOP value is lower than any OCR1A/B/C register, a compare match will never occur.

In this mode the compare units generates PWM waveforms on the OC1A/B/C pins. The PWM waveform is generated by setting (or clearing) the OC1A/B/C register at the compare match between OCR1A/B/C and TCNT1 and clearing (or setting) the OC1A/B/C register at the timer clock cycle the counter is cleared. The PWM frequency for the output waveform can be calculated using the following equation.

$$f_{OCnxPWM} = \frac{f_{clkI/O}}{Prescaler \bullet (1 + TOP)}$$

The frequency range analysis in the Fast PWM mode is provided below.

Table 4: Frequency Range Analysis

TOP	Frequency Hz (Prescaler 1)	Frequency Hz (Prescaler 8)	Frequency Hz (Prescaler 64)	Frequency Hz (Prescaler 256)	Frequency Hz (Prescaler 1024)
0	16000000	2000000	250000	62500	15625
3277	4881.025	610.1281	76.26602	19.0665	4.766626
6554	2440.885	305.1106	38.13883	9.534706	2.383677
9831	1627.339	203.4174	25.42718	6.356794	1.589199
13108	1220.536	152.5669	19.07087	4.767717	1.191929
16385	976.4433	122.0554	15.25693	3.814232	0.953558
19662	813.711	101.7139	12.71423	3.178559	0.79464
22939	697.4717	87.18396	10.89799	2.724499	0.681125
26216	610.291	76.28638	9.535797	2.383949	0.595987
29493	542.4832	67.8104	8.4763	2.119075	0.529769
32770	488.2366	61.02957	7.628696	1.907174	0.476794
36047	443.8526	55.48158	6.935198	1.733799	0.43345
39324	406.8659	50.85823	6.357279	1.58932	0.39733
42601	375.5692	46.94615	5.868269	1.467067	0.366767
45878	348.7434	43.59293	5.449116	1.362279	0.34057
49155	325.4943	40.68679	5.085849	1.271462	0.317866
52432	305.1513	38.14392	4.76799	1.191997	0.297999
55709	287.2016	35.9002	4.487525	1.121881	0.28047
58986	271.2462	33.90578	4.238222	1.059555	0.264889
62263	256.9703	32.12129	4.015161	1.00379	0.250948

Setting the OCR1A/B/C register to 0x00 will result in narrow spike for teach TOP+1 timer clock cycle and if it set to the TOP then the output will be constant high or low depending on the polarity of the output set by the COM1x1:0 bits.) The following table shows how COM1x1:0 bits can be configured for desired compare output modes in Fast PWM.

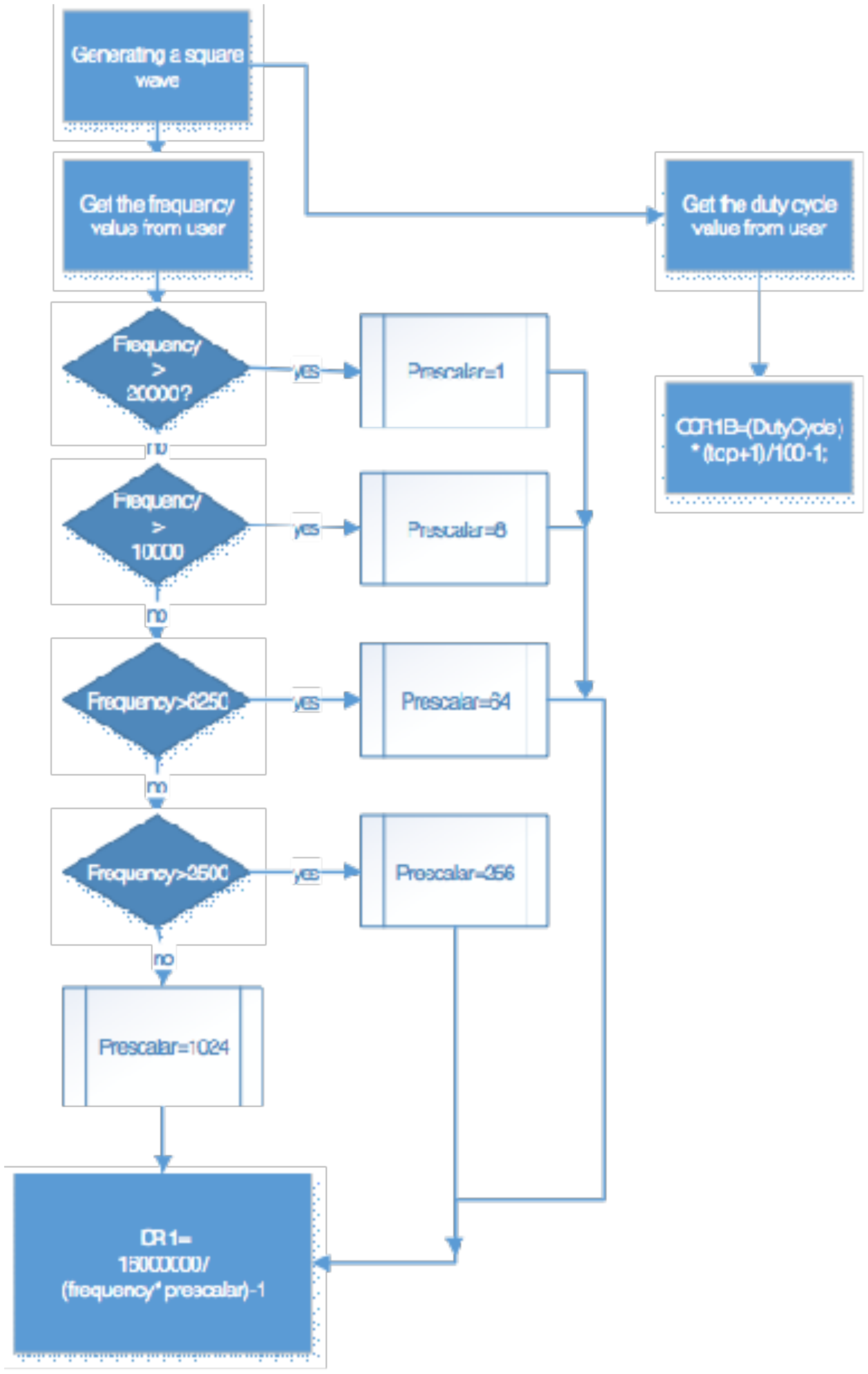
Table 5: Compare Output Mode, Fast PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGMn3:0 = 15: Toggle OCnA on Compare Match, OCnB/OCnC disconnected (normal port operation). For all other WGMn settings, normal port operation, OCnA/OCnB/OCnC disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM, (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM, (inverting mode)

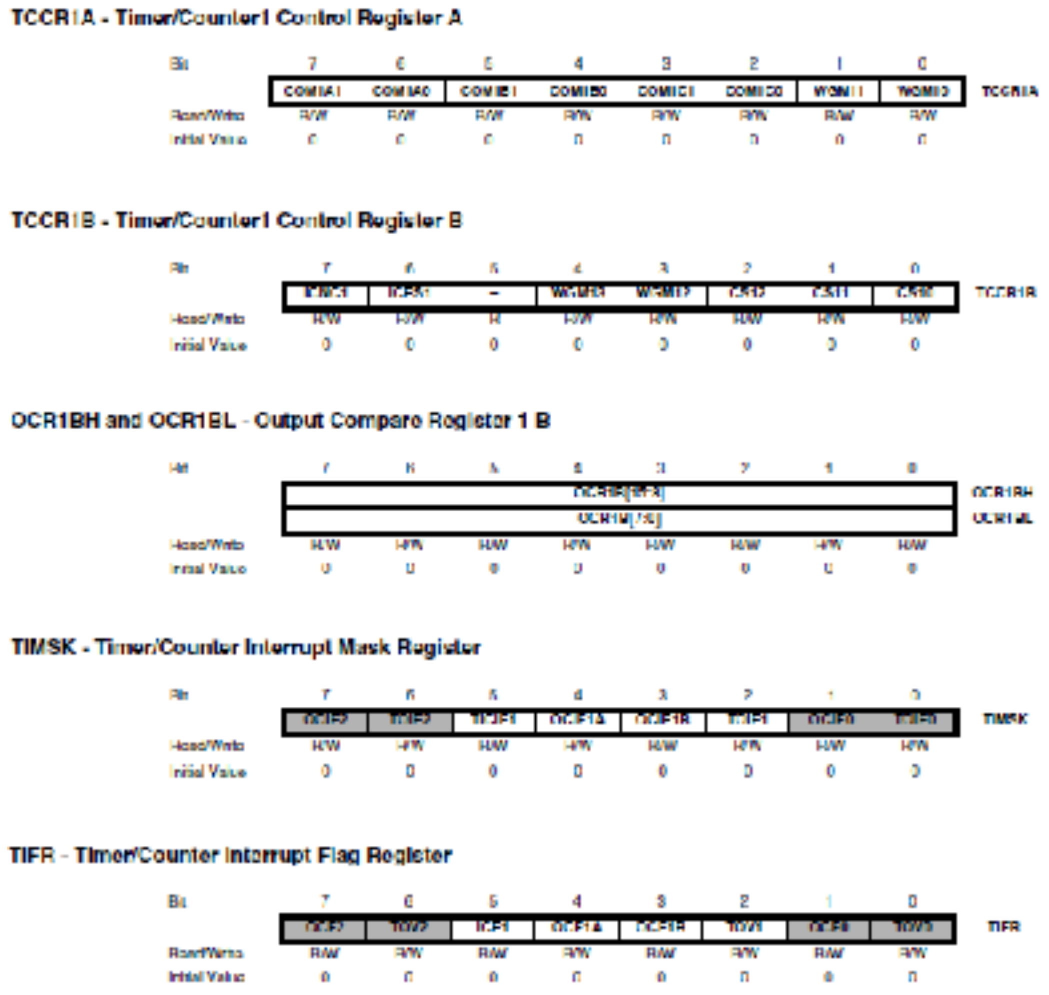
The following code snippet sets up the ATmega128 to generate square waves of variable duty cycle.

```
int main( void )
{
    ICR1=65535;                      //set the top value to be max
    OCR1B = 2;                  // Initialize Output Compare Register to allow
                                 // 100% duty cycle
    //DDRB = (1<<PB6);               // set OC1 to output
    TCNT1 = 0;                     // start the counter at zero
    TCCR1A = (1<<COM1B1)|(1<<WGM11);  //Fast PWM mode  and non-inverting
output
    TCCR1B=(1<<CS10)|(1<<WGM13)|(1<<WGM12);      // no prescaling



    MCUCR = 0x30;      // Sleep enabled for power down mode.
    EIMSK = 0x01;      // Enable interrupt INT0.
                     // Low level by default (EICRA not written)
    while (1) {
  }
}
```

Section 2.4.5: Timer1 Square-wave Software Description



The following tables provide brief descriptions and locations of some of the register, clock select bits, waveform generation mode bits and compare output mode bits.



TCCR1A - Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1B - Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

OCR1BH and OCR1BL - Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TIMSK - Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TIFR - Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 19: Description of Control Bits (Source: ATmega128 datasheet)

Section 2.5: AD9834 Direct Digital Synthesis Device

The AD9834 is a low power 75MHz Direct Digital Synthesis device used for generating sinusoidal waves. It is also capable of producing triangular and square waves, but these functions are not used in the final product. The functional block diagram of the device is shown below.

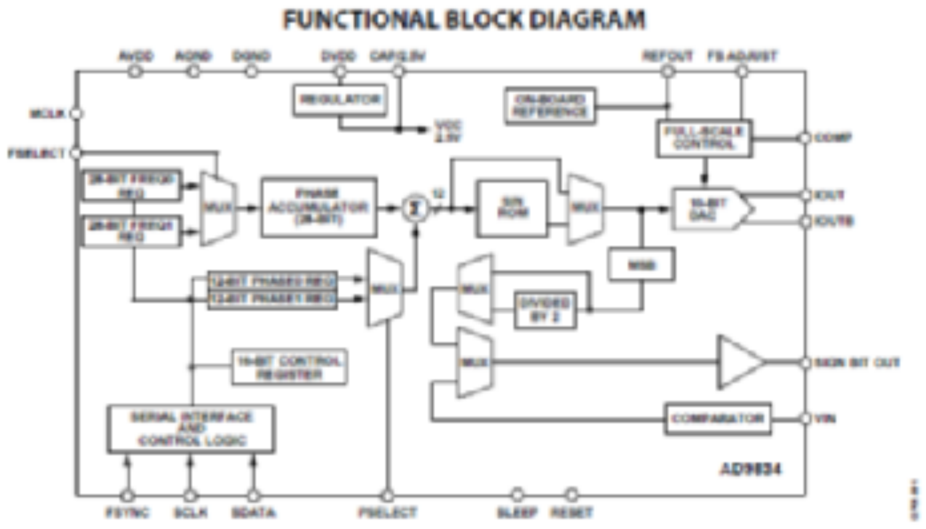


Figure 20: AD9834 Functional Block Diagram (Source: AD9834 datasheet)

Section 2.5.1: Direct Digital Synthesis

Direct Digital Synthesis (DDS) is an efficient method of creating sinusoidal waves of different frequencies based on a single reference frequency. This is a digitally controlled sampled data system which is more efficient at generating sinusoidal waves compared to the ATmega128. However, the ATmega128 is more suitable for generating square-waves by using pulse-width modulation.

The architecture of a basic DDS system consists of a stable clock that drives a programmable-read-only-memory (PROM) which contains one or more integral number of cycles of an arbitrary waveform (in this case a sinewave). As the address counter is incremented, the corresponding digital amplitude of the signal at each address drives a DAC. This DAC in turn produce an analog output signal. A fundamental problem with a basic DDS system is that the output frequency can only be changed by changing the reference clock frequency or by reprogramming the PROM; both of which is time consuming and inflexible. A practical DDS implements this function using Numerically Controlled Oscillator (NCO). The block diagram of such a system is provided below.

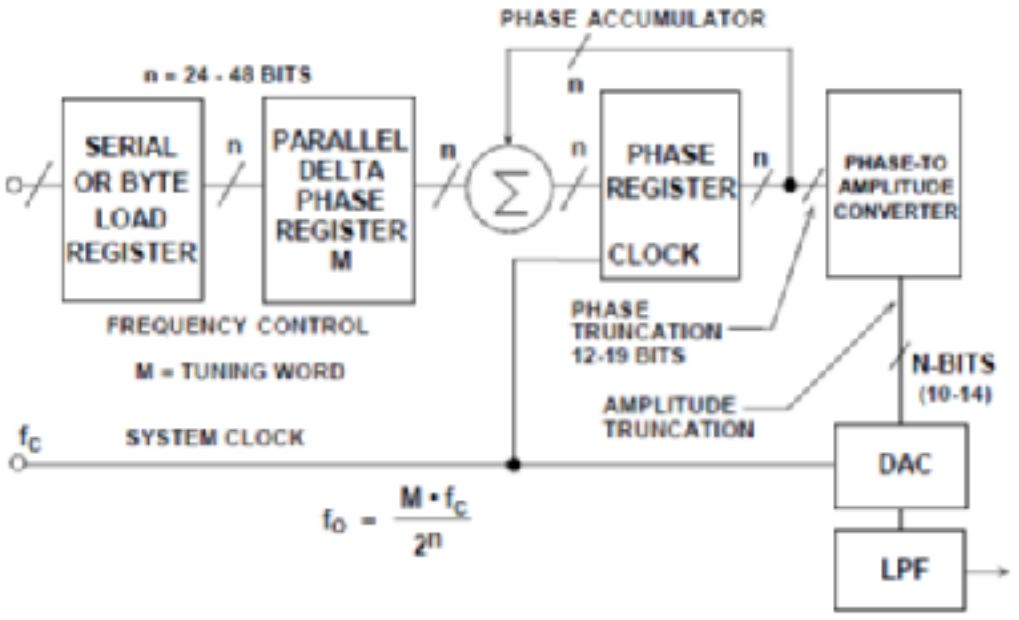
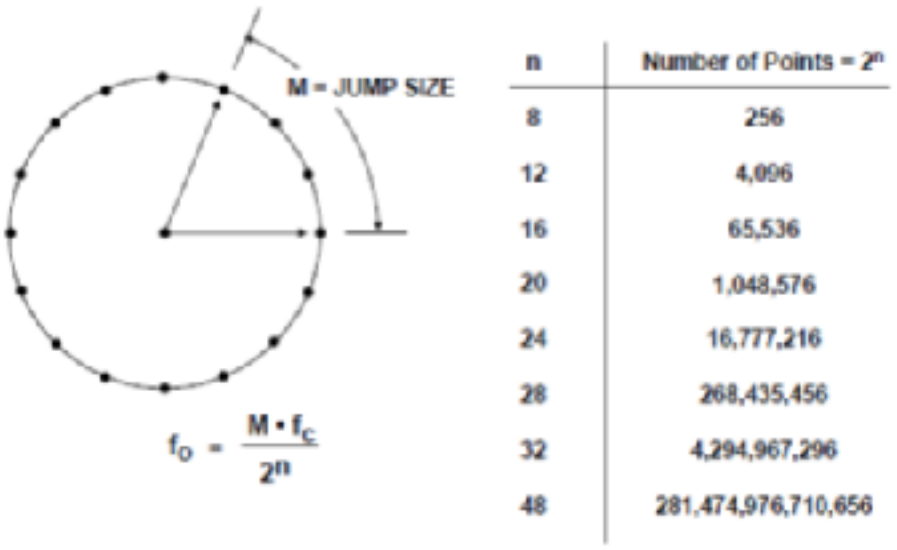


Figure 21: A Flexible DDS System (Source: Fundamental of DDS MT-085)

The content of the phase accumulator, shown in the figure above, is updated once each clock cycle. During the update process the number, **M**, stored in the delta phase register is added to the content of the phase accumulator register. If the number stored in the phase accumulator register is **m** and the number stored in the delta phase register is **M** then the phase accumulator is update by **m = m + M**. If the accumulator is **n** bits wide then 2^n cycles are required before the phase accumulator returns to 00…00, and the cycle is repeated. The output of the phase accumulator is the address to a sine (or cosine) lookup table. Each address in this lookup table contains the digital amplitude in formation for one complete cycle of a sinewave although; in a practical DDS system only the data for the first 90 degrees are used. This information is then used to drive the DAC. The following diagram provides a visual representation of the operation.



n	Number of Points = 2^n
8	256
12	4,096
16	65,536
20	1,048,576
24	16,777,216
28	268,435,456
32	4,294,967,296
48	281,474,976,710,656

Figure 22: Digital Phase Wheel (Source: Fundamental of DDS MT-085)

The following equation can be used to calculate the output frequency $f_o = \frac{M \times f_c}{2^n}$. If **M** is higher than 1 then the output frequency will be **M** time faster. This equation is also known as the “tuning equation.” The frequency resolution in this case will be $\frac{f_c}{2^n}$. In practical DDS system only the first 13 to 15 MSBs are passed on to the lookup table. This does not affect the frequency resolution and only adds a small amount of phase noise to the output. Since any N-bit DAC will add quantization noise to the output this small amount of phase noise is acceptable. The frequency range analysis of AD9834 is given below.

Table 6: Frequency Range Analysis for AD9834

Delta Phase Register M	Frequency Hz
0	0
205	2502441
410	5004883
615	7507324
820	10009766
1025	12512207
1230	15014648
1435	17517090
1640	20019531
1845	22521973
2050	25024414
2255	27526855
2460	30029297
2665	32531738
2870	35034180
3075	37536621
3280	40039063
3485	42541504
3690	45043945
3895	47546387

The calculations required to determine the value of M is performed in the function Set_frequency_register(long frequency, char freqSelect) which is described in the software section later.

The following figure shows the calculated output spectrum for 15-bit phase truncation. The spurs caused by the truncation are 90 dB below the full scale output.

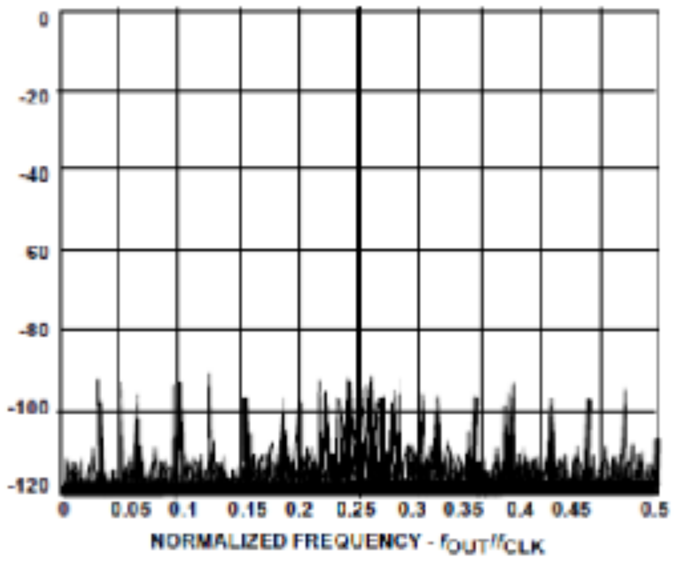
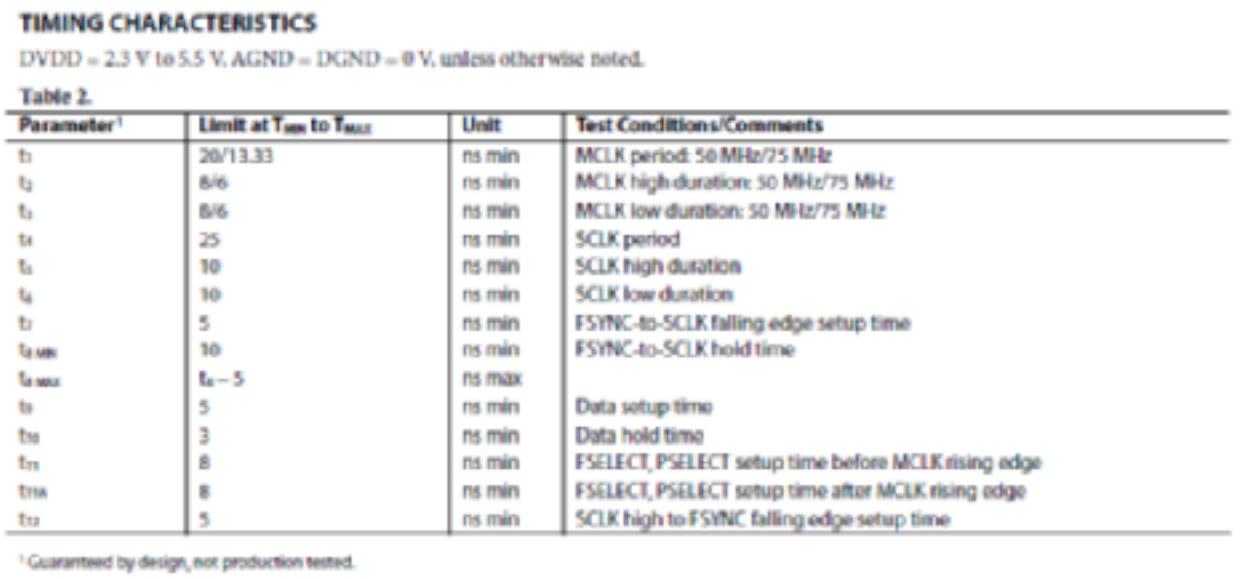


Figure 23: Output Spectrum (Source: Fundamental of DDS MT-085)

The internal buffer register that precedes the parallel-output delta phase register is used to change all the bits of the delta phase register simultaneously. The number of clock cycles needed to load the delta phase register determined the maximum rate at which the output frequency can be changed. Since DDS system is a sampled data system, all the relevant issues such as aliasing, filtering etc. involved in sampling must be considered.

Section 2.5.2 AD9834 Hardware Description

This DDS system consists of two frequency select registers and a 28-bit phase accumulator. The output of the NCO, in this case, is truncated to 12 bits. The input to the phase accumulator can be selected either from the FREQ0 or FREQ1 register. The content of the either frequency select register can be transmitted by a standard 3-wire serial interface. The data is loaded into the device as a 16-bit word under the control of a serial clock input (SCLK). The timing diagram for this operation is shown below.



TIMING CHARACTERISTICS

DVDD = 2.3 V to 5.5 V, AGND = DGND = 0 V, unless otherwise noted.

Table 2.

Parameter[1]	Limit at T_{MIN} to T_{MAX}	Unit	Test Conditions/Comments
t_1	20/13.33	ns min	MCLK period: 50 MHz/75 MHz
t_2	8/6	ns min	MCLK high duration: 50 MHz/75 MHz
t_3	8/6	ns min	MCLK low duration: 50 MHz/75 MHz
t_4	25	ns min	SCLK period
t_5	10	ns min	SCLK high duration
t_6	10	ns min	SCLK low duration
t_7	5	ns min	FSYNC-to-SCLK falling edge setup time
$t_{8\,MIN}$	10	ns min	FSYNC-to-SCLK hold time
$t_{8\,MAX}$	$t_4 - 5$	ns max	
t_9	5	ns min	Data setup time
t_{10}	3	ns min	Data hold time
t_{11}	8	ns min	FSELECT, PSELECT setup time before MCLK rising edge
t_{11A}	8	ns min	FSELECT, PSELECT setup time after MCLK rising edge
t_{12}	5	ns min	SCLK high to FSYNC falling edge setup time

[1] Guaranteed by design, not production tested.

Timing Diagrams

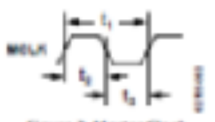


Figure 3. Master Clock

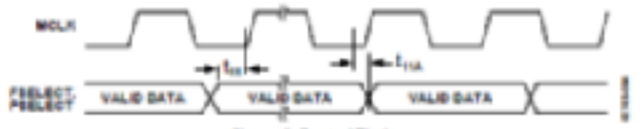


Figure 4. Control Timing

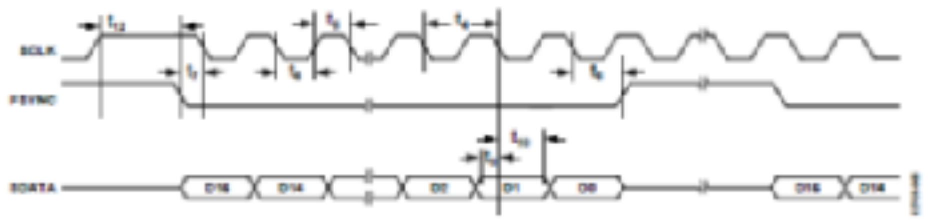


Figure 5. Serial Timing

Figure 24: Timing Characteristics of AD9834 (Source: AD9834 datasheet)

A 16-bit control register is used to setup the AD9834. To alter the control register the two MSBs (bit 15 and 14) must be set to zero before the data is passed to the AD9834. The rest of the bits are used to configure the device. A table explaining the function of each bit in in the control register is given the appendix A1. The following program shows how to setup the master ATmega128 to transfer data to AD9834. In this case PB1, PB2 and PG0 are used as SCK, MOSI and /SS respectively.

```
//Initialization Routine Master Mode(Polling)
void Init_Master(void)
{
    volatile char IOReg;
    //set PB0(/SS),PB1(SCK),PB2(MOSI) as output
    IOReg=DDRB;
    DDRB=IOReg|(1<<PB0)|(1<<PB1)|(1<<PB2);      //PB 0,1,2 are used for seri-
al
                                                //transfer
    //unassert LCD slave
    SETBIT(PORTB,0);
    //enble SPI in Master Mode with SCK=CLK/4
    DDRG=0x01;  //PORTG0 as output to /SS
    PORTG=0x01; //deselect /SS initially
    //idle level is 1 and it samples at leading edge
    SPCR=(1<<SPE)|(1<<MSTR)|(1<<CPOL)|(0<<CPHA);
    SPCR |=0x01;
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
    IOReg=SPDR;

}
```

The following code snippet configures the control register of the AD9834 to use the 28-bit frequency register FREQ0 to update the phase accumulator and to update FREQ0 in two consecutive writes.

```
//set B28 to allows a complete word to be loaded into a frequency register in
//two consecutives
void Init_AD9834(void){

      volatile char IOReg, sendbyte;
      volatile int CONTROLReg;

      CONTROLReg = (1<<B28) ;     //config contrlreg to load the 28-bit freq
                                  //registers in 2 consecutive writes
      sendbyte = (CONTROLReg >>8);                  //get high byte

      CLEARBIT(PORTG,0);        //assert ss for AD9834

      PORTG=0x00;
      SPDR = sendbyte;          //send high byte
      while (!(SPSR & (1<<SPIF)));    // wait until Char is sent
      IOReg   = SPSR;                   // clear SPIF bit in SPSR
      IOReg   = SPDR;

      sendbyte = CONTROLReg & 0x00FF;           //get low byte

      SPDR = sendbyte;          //send low byte
      while (!(SPSR & (1<<SPIF)));    // wait until Char is sent
      IOReg   = SPSR;                   // clear SPIF bit in SPSR
      IOReg   = SPDR;

      SETBIT(PORTG, 0);         //unassert ss for AD9834

}
```

This product does not use the phase registers of the AD9834, but in order to select the appropriate frequency registers the two MSBs of the 16-bit data must be setup appropriately according to the following table.

Table 7: Frequency Register Bits (Source: AD9834 datasheet)

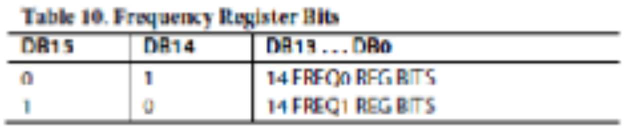


Table 10. Frequency Register Bits

DB15	DB14	DB13 . . . DB0
0	1	14 FREQ0 REG BITS
1	0	14 FREQ1 REG BITS

Since both FREQ0 and FREQ1 registers are 28-bit wide and the SPI Data Register of ATmeaga128 is only 8-bit wide; four consecutive writes to the same address must be performed to update the frequency registers. The 8 MSBs of the lower word (16-bits) must be written first and the 8 LSBs of the lower word must follow. The same pattern must be repeated for the higher word as well. Each time the 16-bit data is transmitted the 2 MSBs of the transmitted byte must be either 01 or 10 depending on whether FREQ0 or FREQ1 register is updated. It is also possible to update only the 14 MSBs or the 14 LSBs in order to fine tune the output. It can be done by setting the 13th bit of the control register to 0. The following table provides some examples.

Table 8: Writing to Frequency Registers (Source: AD9834 datasheet)

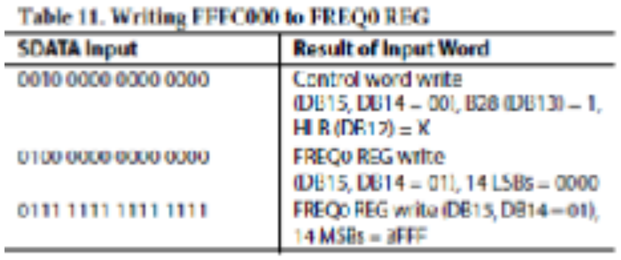


Table 11. Writing FFFC000 to FREQ0 REG

SDATA Input	Result of Input Word
0010 0000 0000 0000	Control word write (DB15, DB14 = 00), B28 (DB13) = 1, HLB (DB12) = X
0100 0000 0000 0000	FREQ0 REG write (DB15, DB14 = 01), 14 LSBs = 0000
0111 1111 1111 1111	FREQ0 REG write (DB15, DB14 = 01), 14 MSBs = 3FFF

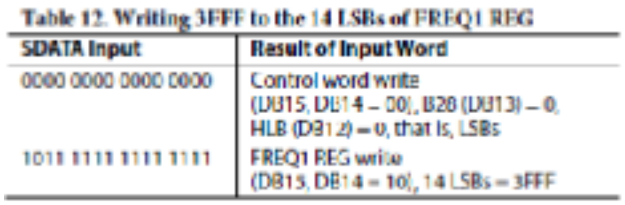


Table 12. Writing 3FFF to the 14 LSBs of FREQ1 REG

SDATA Input	Result of Input Word
0000 0000 0000 0000	Control word write (DB15, DB14 = 00), B28 (DB13) = 0, HLB (DB12) = 0, that is, LSBs
1011 1111 1111 1111	FREQ1 REG write (DB15, DB14 = 10), 14 LSBs = 3FFF

Section 2.5.3: AD9834 Software Description

The following snippet of code shows how to update the FREQ0 register in 4 consecutive writes. This function also calculates the delta phase register value for the user defined frequency. The underlying mathematics of the calculation is described in the Direct Digital Synthesis section which was presented previously.

```
//set the frequency
//select the frequency register to alter
void Set_frequency_register(long frequency,char freqSelect)
{
    volatile char IOReg,copyOfDelta1,copyOfDelta2,copyOfDelta3,copyOfDelta4;
    volatile long delta;
    //Fout=fmclk/268435456*FREQREG
    delta=268435456/50000000;
    delta=delta*frequency;

    //assert /SS
    PORTG=0x00;

    //send bits 8 to 13 and select the frequency register to alter
    //choose the frequency register based on the parameter
     copyOfDelta2=(delta>>8);

     if(freqSelect==0)
      copyOfDelta2=copyOfDelta2|(1<<(DB14-8))|(0<<(DB15-8));
    else if(freqSelect==1)
      copyOfDelta2=copyOfDelta2|(0<<(DB14-8))|(1<<(DB15-8));

//    copyOfDelta1 = delta;
     SPDR=copyOfDelta2;
    //wait for the transmission to complete
    while(!TESTBIT(SPSR,SPIF));
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
    IOReg=SPDR;

    //send bits 0 to 7
    copyOfDelta1 = delta;
    SPDR=copyOfDelta1;
    //wait for the transmission to complete
    while(!TESTBIT(SPSR,SPIF));
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
    IOReg=SPDR;
    //shift to right for 8 positions

     //send bits 22 to 27 and select the frequency register to alter
    //choose the frequency register based on the parameter
    copyOfDelta4=(delta>>22);

    if(freqSelect==0)
      copyOfDelta4=copyOfDelta4|(1<<(DB14-8))|(0<<(DB15-8));
    else if(freqSelect==1)
      copyOfDelta4=copyOfDelta4|(0<<(DB14-8))|(1<<(DB15-8));

    SPDR=copyOfDelta4;
    //wait for the transmission to complete
    while(!TESTBIT(SPSR,SPIF));
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
    IOReg=SPDR;

       //send bits 14 to 21
    copyOfDelta3=delta>>14;
    SPDR=(copyOfDelta3);
    //wait for the transmission to complete
    while(!TESTBIT(SPSR,SPIF));
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
    IOReg=SPDR;

     //unassert the slave unit last
     PORTG=0x01;
}
```

The initialization, data write and operation of the AD9834 can be summarized using the following flowcharts.

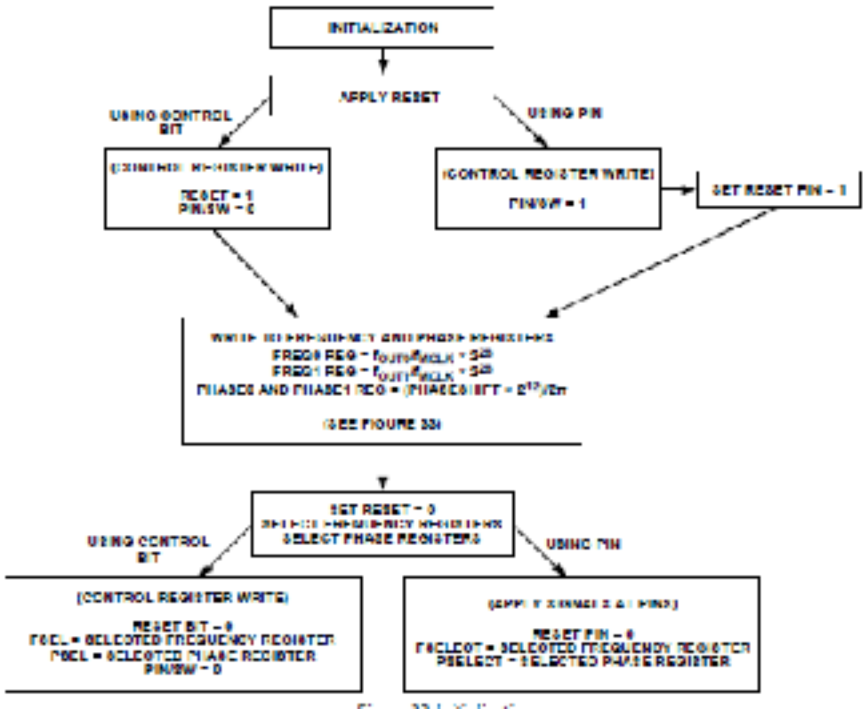

INITIALIZATION
APPLY RESET
USING CONTROL BIT
USING PIN
(CONTROL REGISTER WRITE)
RESET = 1
PIN/SW = 0
PIN/SW = 1
SET RESET PIN = 1
WRITE TO FREQUENCY AND PHASE REGISTERS
(SEE FIGURE 33)
SET RESET = 0
SELECT FREQUENCY REGISTERS
SELECT PHASE REGISTERS
RESET BIT = 0
FSEL = SELECTED FREQUENCY REGISTER
PSEL = SELECTED PHASE REGISTER
RESET PIN = 0
FSELECT = SELECTED FREQUENCY REGISTER
PSELECT = SELECTED PHASE REGISTER

Figure 25: Initialization (Source: AD9834 datasheet)

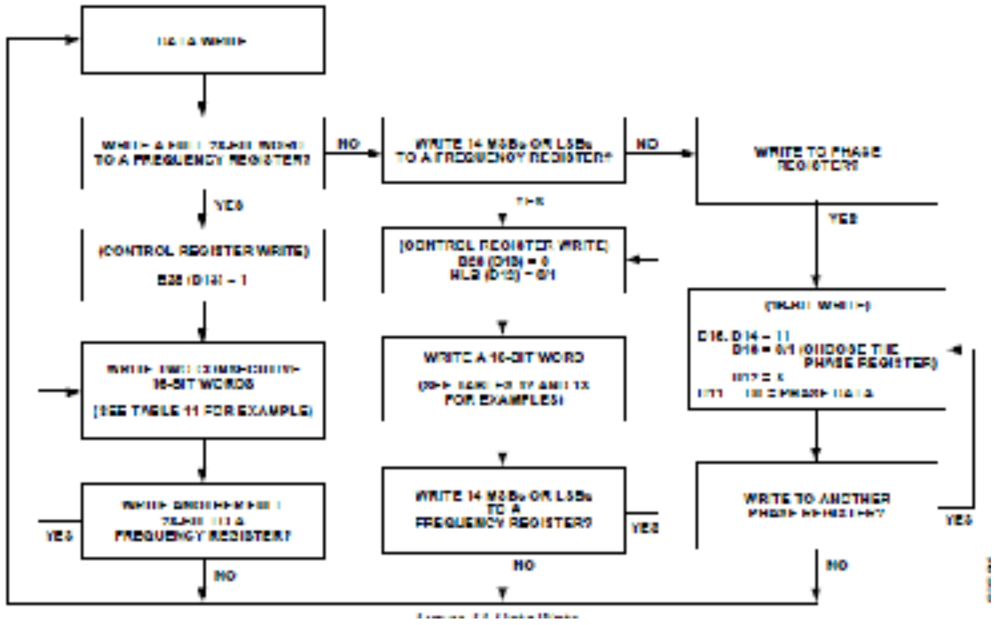


Figure 26: Data Write (Source: AD9834 datasheet)

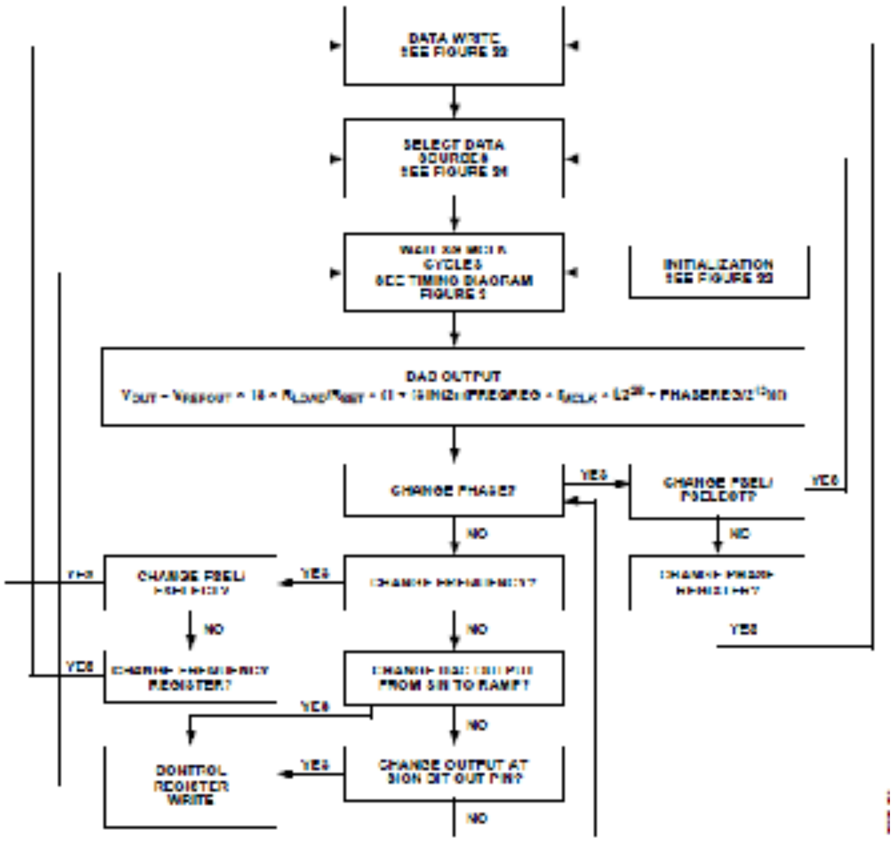

DATA WRITE SEE FIGURE 28
SELECT DATA SOURCES SEE FIGURE 31
WAIT 8/9 MCLK CYCLES SEE TIMING DIAGRAM FIGURE 1
INITIALIZATION SEE FIGURE 28
DAC OUTPUT
CHANGE PHASE?
CHANGE PSEL/PSEL OPT?
CHANGE PHASE REGISTER?
CHANGE FSEL/FSELECT?
CHANGE FREQUENCY?
CHANGE FREQUENCY REGISTER?
CHANGE DAC OUTPUT FROM SIN TO RAMP?
CHANGE OUTPUT AT SIGN BIT OUT PIN?
CONTROL REGISTER WRITE
YES
NO

Figure 27: Flowchart for Initialization and Operation (Source: AD9834 datasheet)

Section 2.6: MAX5402 Digital Potentiometer

The MAX5402 is a digital potentiometer with a 256-tap variable resistor with a total resistance of 10kΩ. The device is capable of providing glitch less switching between resistor taps and midscale power-on reset. Although digital potentiometers can be used as adjustable voltage references and current to voltage converters; the MAX5402 is used as an adjustable gain amplifier in this product. The digital potentiometer shares the same architecture as a string digital to analog converter. A brief description of string digital to analog converters is presented before the hardware description of the MAX5402 is provided.

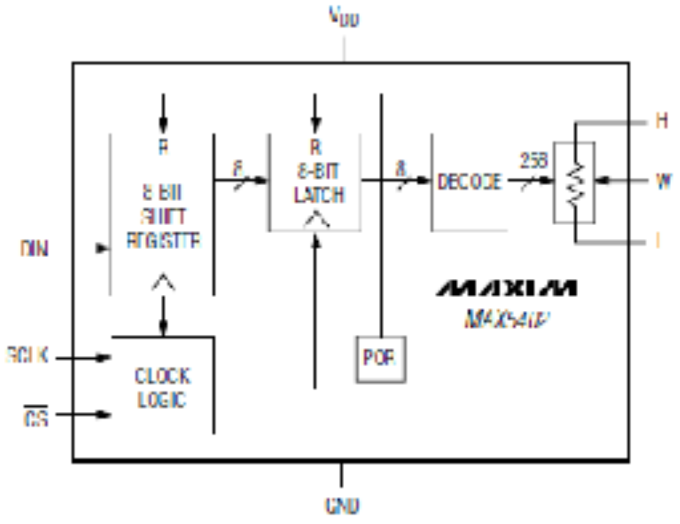


Figure 28: MAX5402 Functional Diagram (Source: MAX5402 datasheet)

Section 2.6.1: String Digital to Analog Converter (DAC)

A string DAC, also known as a Kelvin DAC, has the simplest structure of all DACs. An n-bit version of a string DAC consists of 2^n resistors of equal values in series with 2^n CMOS switches that are connected to the output. A simple diagram of a string DAC is given below.

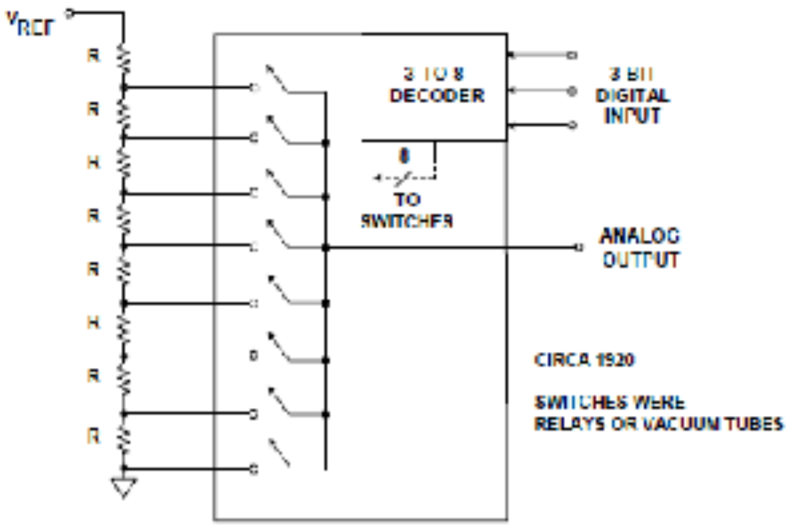


Figure 29: String DAC (Source: Basic DAC Architectures by Walt Kester)

The 3-to-8 decoder is used to close the appropriate switch so that the output can be taken from the appropriate tap. This DAC is inherently monotonic, that is even if a resistor is accidentally short-circuited, and the output n cannot exceed output n+ 1. The DAC can be easily transformed between linear and nonlinear by varying the resistor values – the DAC is linear if all the resistors are equal. Also, the switching glitch in string DAC is not code dependent and the glitch is relatively constant regardless of code transition. As such, string DACS are ideal for low distortion application. The major drawback of the string DAC is the large number of resistors and switches required for high resolution. However, with the advent of small IC this challenge has been overcome.

Section 2.6.2: Digital Potentiometer

The operation of a digital potentiometer is same as the string DAC. The major difference between the two is that the digital potentiometer lacks a resistor and neither end of the string of resistors has any internal connection. The diagram of a simple digital potentiometer is shown below.

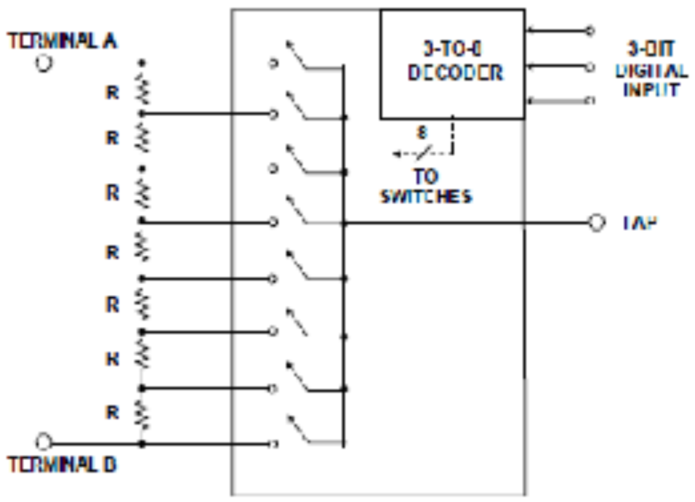


Figure 30: Digital Potentiometer (Source: Basic DAC Architectures by Walt Kester)

Similar to the string DAC, the 3-to-8 decoder is used to close appropriate switches (two at a time) to create a variable resistance. Digital potentiometers often incorporate persistent (nonvolatile) logic so their settings are retained when they are turned off.

Section 2.6.3: MAX5402 Hardware Description

The MAX5402 consists of 255 fixed resistors in series between the High (H) and Low (L) pins. The wiper (W) is programmed to access any one of the 256 different tap points on the resistor string. The wiper tap position can be controlled using an SPI-compatible 3-wire serial data interface. This is a write only interface and contains chip select (/CS), data in (DIN) and data clock (SCLK) inputs. To load data into the 8-bit shift register the /CS pin must be taken low and maintained until data transfer is over. Data is loaded on the rising of each SCLK pulse and the MSB is shifted in first. After all the 8 bits have been loaded into the shift register they are latched into the decoder when /CS is taken high. The decoder then switches the wiper to the tap position corresponding to the 8-bit input data. The resistance corresponding to a decimal input DEC can be determine using the following equation.

$$DEC \times \frac{10k\Omega}{255}$$

The MAX5402 sets the wiper to midscale position at power-up by loading a binary value of 128 into the 8-bit latch. The serial interface timing diagrams for MAX5402 is shown below.

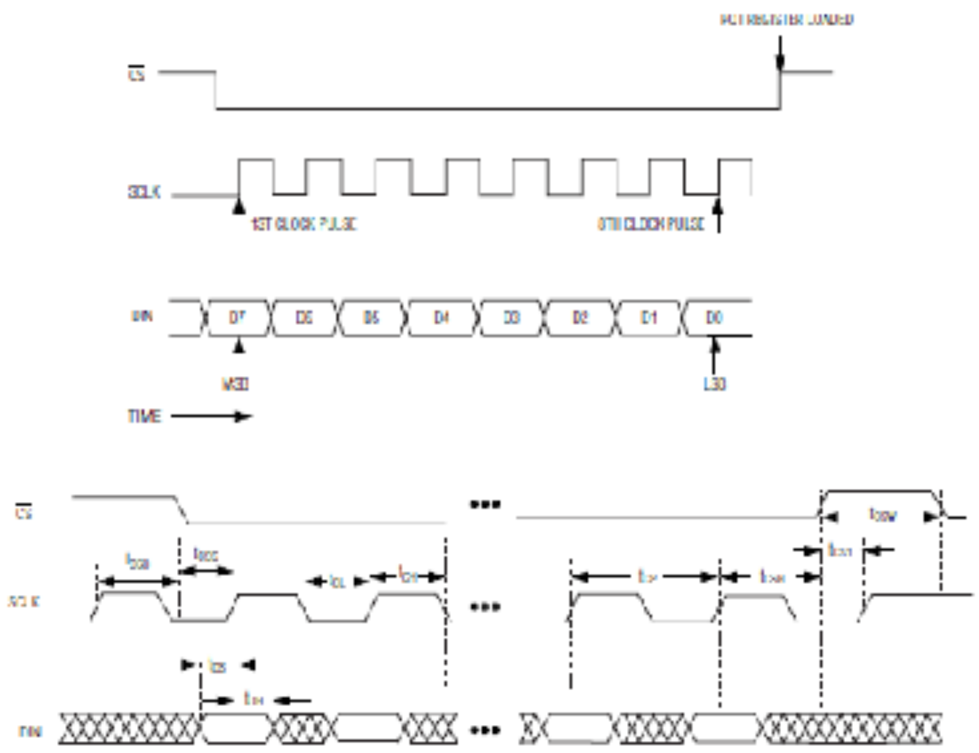


Figure 31: Serial Interface Timing Diagram (Source: MAX5402 datasheet)

Section 2.6.4: MAX5402 Software Description

The following code snippet shows how to setup the master, ATmega128 to transfer data to MAX5402.

```
//Initialization Routine Master Mode(Polling)
void Init_Master_MAX5402(void)
{
    volatile char IOReg;
    //set PB0(/SS),PB1(SCK),PB2(MOSI) as output
    //unassert LCD slave
    SETBIT(PORTB,0);
    //unassert DDS slave
    IOReg=DDRB;
    DDRA=0xFF;
    PORTA=0xFF;
    DDRB=IOReg|(1<<PB0)|(1<<PB1)|(1<<PB2);
    //unassert LCD slave
    SETBIT(PORTB,0);
    //enble SPI in Master Mode with SCK=CLK/4
    //idle level is 0 and it samples at leading edge
    SPCR=(1<<SPE)|(1<<MSTR)|(0<<CPOL)|(0<<CPHA)|(1<<SPR1);
    SPSR=(1<<SPI2X);
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
    IOReg=SPDR;

}
```

The following code snippet loads an 8-bit data to the MAX5402.

```
void Setup_MAX5402(unsigned char R)
{
    volatile char IOReg;


      CLEARBIT(PORTA,0);          //assert ss for MAX5402

      SPDR = R;            //send byte
      while (!(SPSR & (1<<SPIF)));    // wait until Char is sent
      IOReg   = SPSR;                   // clear SPIF bit in SPSR
      IOReg   = SPDR;

      SETBIT(PORTA, 0);          //unassert ss for MAX5402
}
```

Section 2.7: LM6171 Op-Amp

For a function generator to be useful as a laboratory instrument the output voltage must be larger and adjustable. The AD9834 can output current only and needs to be converted to voltage. In this design a 200Ω resistor and an adjustable gain amplifier is used to convert the current waveform to voltage waveform. The LM6171 is a high speed unity-gain voltage feedback amplifier. It is used with the MAX5402 to create the adjustable gain amplifier. The schematic diagram for the adjustable gain amplifier is shown below.

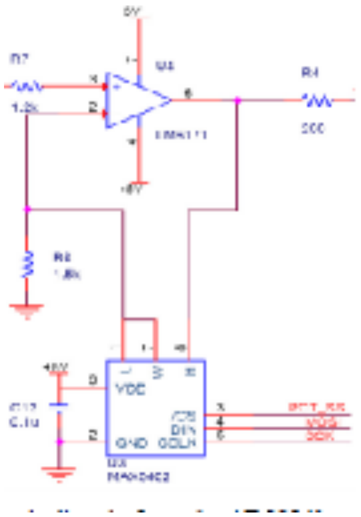


Figure 32: Adjustable Gain Amplifier (Source: Lab Module #5)

An ideal amplifier takes the voltage difference between inverting and non-inverting input and amplifies it by an open loop gain. The open loop gain of each amplifier varies from part to part and as such it is highly unreliable for design consideration. In this design the LM6171 is used as a negative feedback amplifier where the gain is determined by the feedback network that consists of a voltage divider. The output from the AD9834 is fed into the non-inverting input of the LM6171. The inverting input of the op-amp is connected to a voltage divider that consists of 1.5kΩ resistor and the MAX5402 digital potentiometer. The gain of the output can be determined by summing the current at the inverting node of the op-amp. The gain can be calculated using the following equation.

$$V_{out} = (1 + \frac{R_{MAX5402}}{1.5k\Omega}) \times V_{in}$$

The resistance of digital potentiometer can be varies using the microcontroller. The exact method of the varying the resistance is described in digital potentiometer section of this report. The output of the amplifier is then passed through an analog filter to eliminate noise in the signal.

Section 2.8: Serial Peripheral Interface (SPI)

To send instructions and data to the DDS chip AD9834 and the LCD module from ATmega128, Serial Peripheral Interface is used. In this project, ATmega128 is the master unit and the slaves unit will be AD9834 and LCD module. Figure 1 shows the SPI interface between ATmega128 and other three other slaves.

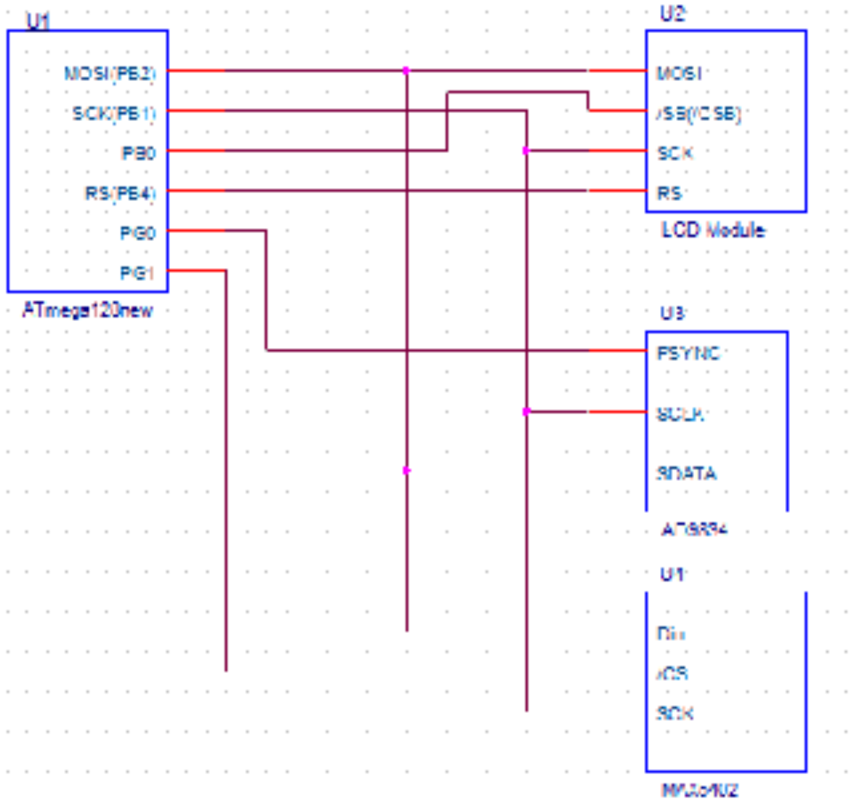


Figure 33: SPI interface between ATmega128 and Slave Units

In Figure 6, PB0 is the slave select for LCD module and PG0 is the slave select for AD9834. All the data and instructions are sent to slave unit from MOSI. SCK from ATmega128 provides the clock signal to the slave unit. Only one or none of the slaves will be selected at a time.

To use the SPI interface, three registers will be used: SPDR, SPCR and SPSR. SPDR is the data register in which the data and instructions are written in. SPCR is the control register in which we could enable the SPI interface and configure the SPI setting. SPSR is the status register in which we use to check the flags. The following code shows an example to configure SPI interface between DDS and ATmega128. The function Init_Master initializes the SPI and configures ATmega128 as 16MHz. The function sendData will simply send the data by writing data into SPDR. When transmission is complete, SPIF bit in SPSR will be set. The flag is cleared by using a dummy variable to read SPSR and then SPDR.

```
//Initialization Routine Master Mode(Polling)
void Init_Master(void)
{
   //dummy variable used to clear SPIF bit
    volatile char IOReg;
    //set PB0(/SS),PB1(SCK),PB2(MOSI) as output
    IOReg=DDRB;
    DDRB=IOReg|(1<<PB0)|(1<<PB1)|(1<<PB2);
    //unassert LCD slave
   SETBIT(PORTB,0);
    DDRG=0x01;  //PORTG0 as output to /SS
   PORTG=0x01; //deselect /SS initially
   //enable SPI in Master Mode with SCK=CLK/4
    //idle level is 1 and it samples at leading edge
    SPCR=(1<<SPE)|(1<<MSTR)|(1<<CPOL)|(0<<CPHA);
    SPCR |=0x01;
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
   IOReg=SPDR;

}

void sendData(char data)
{
   //dummy variable used to clear SPIF bit
    volatile char IOReg;
   //slave /SS
   PORTG=0X00;
  //send bits 0 to 7
    SPDR=data;
    //wait for the transmission to complete
    while(!TESTBIT(SPSR,SPIF));
    //Clear SPIF bit in SPSR
    IOReg=SPSR;
    IOReg=SPDR;
     //deselect /SS
    PORTG=0x01;
}
```

Section 3: System Characteristics

Standard Waveforms: Sine, Triangle, Square

Section 3.1: Sine characteristics

Sine range: 1Hz to 1.4MHz

Amplitude range: 615.680mV to 4.598V

Amplitude at initialization: 2.607V

Input resolution: 615.68mV

Section 3.2: Triangle characteristics:

Triangle range: 1Hz to 200kHz

Amplitude range: 650mV to 4.5V

Amplitude at initialization: 2.607V

Input resolution: 615.68mV

Section 3.3: Square waveform:

Frequency range: 1Hz to 5MHz

Input frequency resolution: 1Hz to 5MHz

Amplitude: 5V

Square waveform duty cycle range: 1% to 99%

Section 4: Troubleshooting

Section 4.1: Operating Checklist

Before returning your unit for service or repair, please check the following items:

Is the function generator inoperative?

- Verify that the ac power cord is connected to the function generator.
- Verify that the Power switch is on.
- Verify that the power-line fuse is good.

Section 4.2: Cleaning

Clean the outside of the function generator with a soft and slightly dampened cloth. Do not use detergent. Do not disassemble to clean inside.

Section 4.3: Electrostatic Discharge (ESD) Precautions

Almost all electrical components can be damaged by electrostatic discharge (ESD) during handling. Component damage can occur at electrostatic discharge voltages as low as 50 volts.

The following guidelines will help prevent ESD damage when servicing the function generator or any electronic device.

- Disassemble the function generator only in a static-free work area.
- Use a conductive work area to dissipate static charge.
- Use a conductive wrist strap to dissipate static charge accumulation.
- Minimize handling.
- Keep replacement parts in original static-free packaging.
- Use only anti-static solder suckers.

WARNING

SHOCK HAZARD. Only service-trained personnel should remove the instrument covers. To avoid electrical shock and personal injury, make sure to disconnect the power cord from the function generator before removing the covers.

Section 4.4: Troubleshooting Hints

This section provides a brief checklist of common failures. The function generators circuits allow troubleshooting and repair with basic equipment such as a multimeter and oscilloscope.

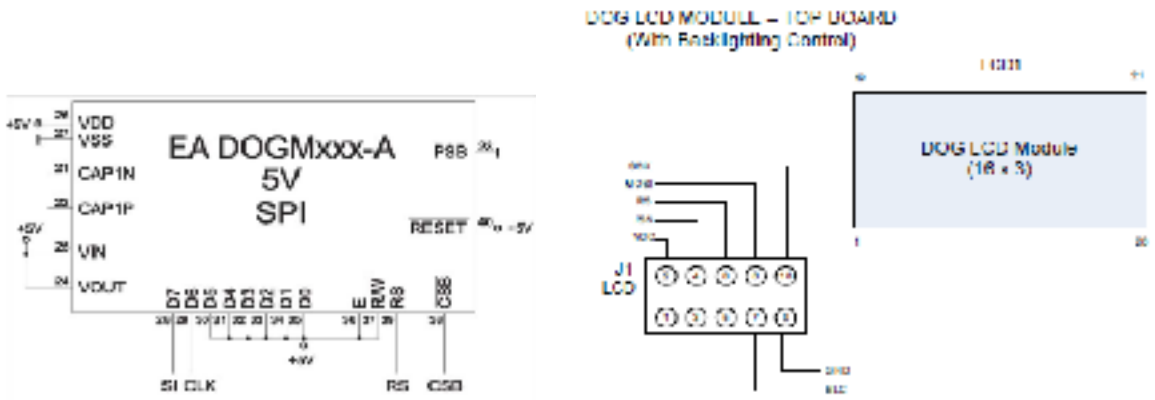


Figure 34: LCD Block Diagram (Source: Lab Module #2)

LCD is Inoperative

- Verify that the voltage at pin 2 of the 10-pin header J1 (from ATmega128 to LCD) is 5V.
- Verify that the voltage at pin 9 is 0V.

Keypad is Inoperative

- Press any key and verify that the voltage at pin PD0 of the ATmega128 is pulled down to 0V.

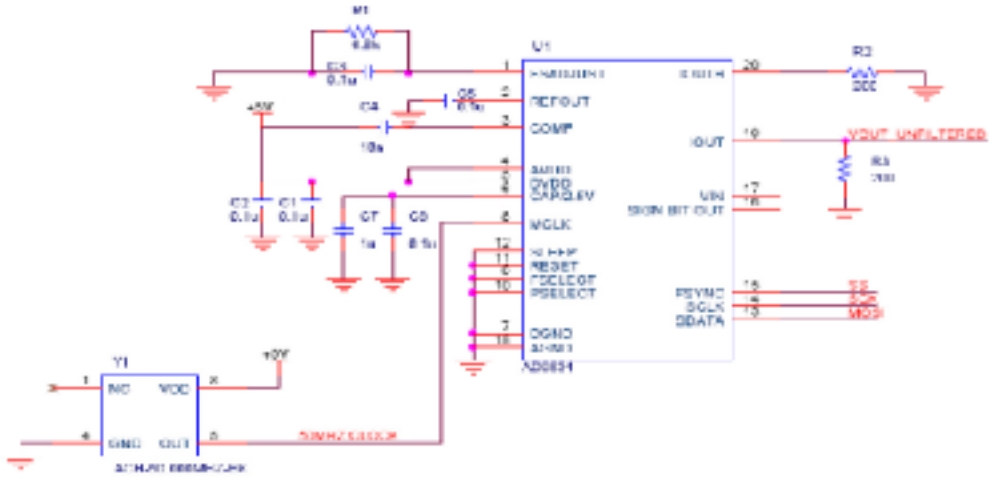


Figure 35: Schematic for AD9834 wiring (Source: Lab Module #4)

AD9834 DDS is Inoperative

- Verify that the external oscillator connected at pin 8, see Figure 32, is running at 50MHz.
- Verify that pin 1 of AD9834 is at 1V.
- Verify that pin 2 is at 1V.
- Verify that pin 3 is at 4V.

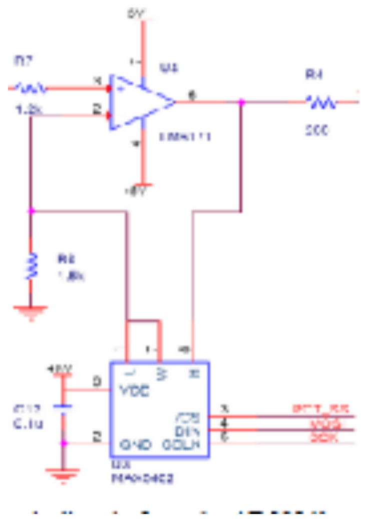


Figure 36: Adjustable Gain Amplifier (Source: Lab Module #5)

MAX5402 Digital Potentiometer is Inoperative

- Verify that the VDD (pin 6 in Figure 33) is at 5V.

Op-Amp is Inoperative

- Verify that input to the op-amp (pin 3 in Figure 33) is a sinusoid or triangular waveform of 550mVp-p with user defined frequency.

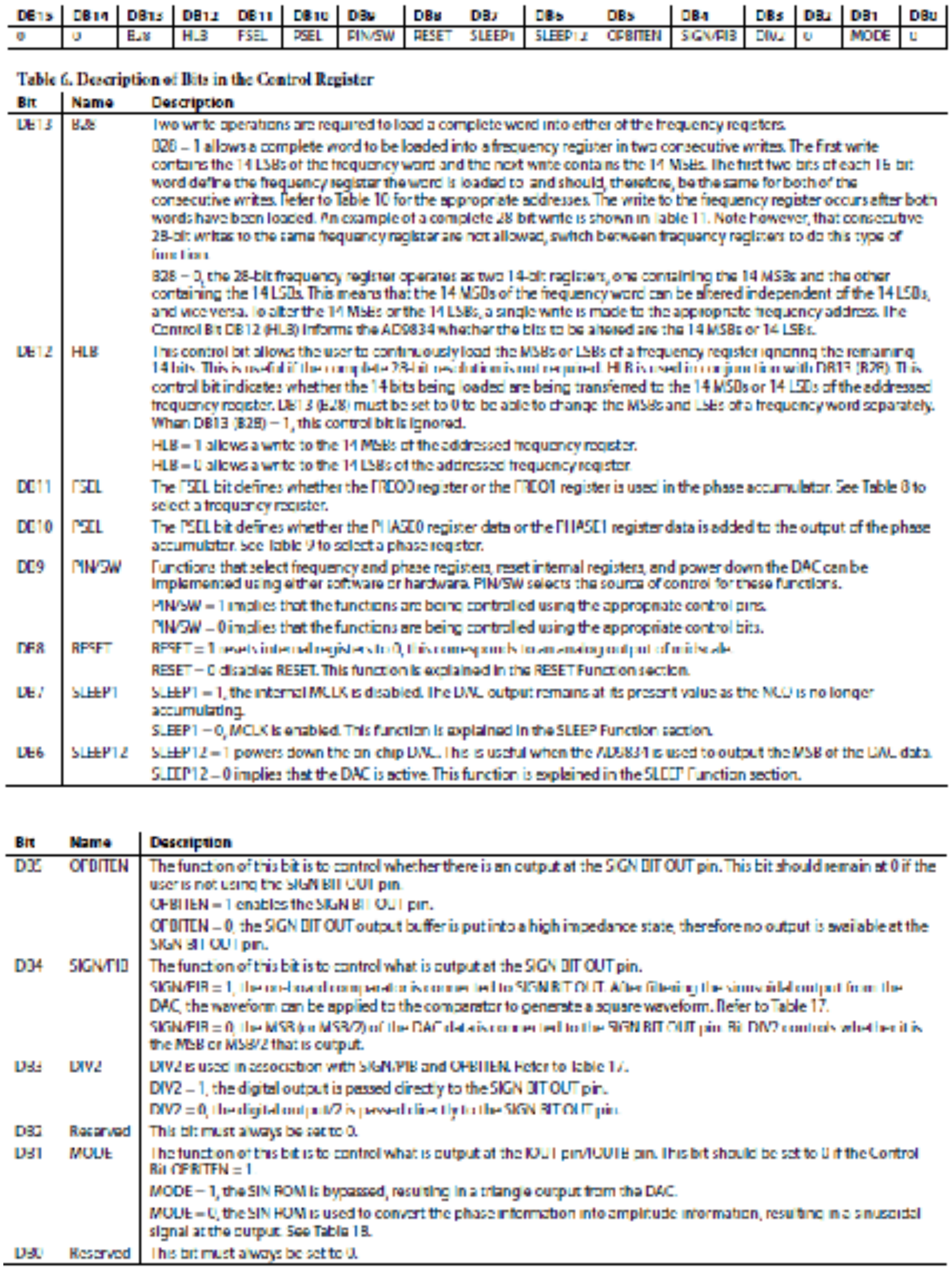
Finally verify the output of the function generator with an oscilloscope: specify a waveform of appropriate frequency, amplitude or duty cycle and monitor the output with the oscilloscope.

Reference

1. ATmega128 Datasheet
2. AVR240: 4X4 Keypad Application note
3. AD9834 DDS Datasheet
4. Fundamentals of DDS MT-085
5. MAX5402 256 TAP Digital Potentiometer Datasheet
6. LM6171 Op-Amp Datasheet
7. ESE 381 Lab Modules #0-5

Appendix

A1: Description of Bits in AD9834 Control Register



DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	B28	HLB	FSEL	PSEL	PIN/SW	RESET	SLEEP1	SLEEP12	OPBITEN	SIGN/PIB	DIV2	0	MODE	0

Table 6. Description of Bits in the Control Register

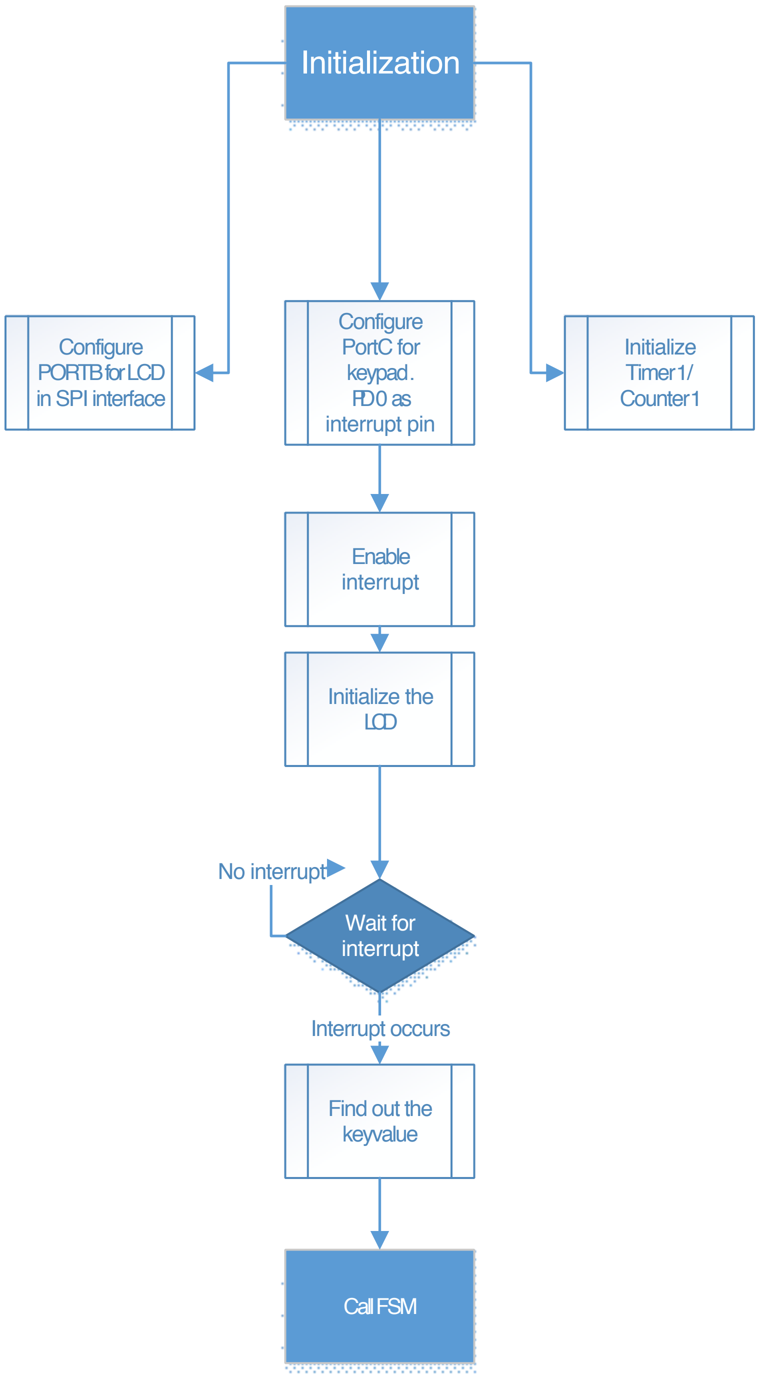
Bit	Name	Description
DB13	B28	Two write operations are required to load a complete word into either of the frequency registers. B28 = 1 allows a complete word to be loaded into a frequency register in two consecutive writes. The first write contains the 14 LSBs of the frequency word and the next write contains the 14 MSBs. The first two bits of each 16-bit word define the frequency register the word is loaded to and should, therefore, be the same for both of the consecutive writes. Refer to Table 10 for the appropriate addresses. The write to the frequency register occurs after both words have been loaded. An example of a complete 28-bit write is shown in Table 11. Note however, that consecutive 28-bit writes to the same frequency register are not allowed, switch between frequency registers to do this type of function. B28 = 0, the 28-bit frequency register operates as two 14-bit registers, one containing the 14 MSBs and the other containing the 14 LSBs. This means that the 14 MSBs of the frequency word can be altered independent of the 14 LSBs, and vice versa. To alter the 14 MSBs or the 14 LSBs, a single write is made to the appropriate frequency address. The Control Bit DB12 (HLB) informs the AD9834 whether the bits to be altered are the 14 MSBs or 14 LSBs.
DB12	HLB	This control bit allows the user to continuously load the MSBs or LSBs of a frequency register ignoring the remaining 14 bits. This is useful if the complete 28-bit resolution is not required. HLB is used in conjunction with DB13 (B28). This control bit indicates whether the 14 bits being loaded are being transferred to the 14 MSBs or 14 LSBs of the addressed frequency register. DB13 (B28) must be set to 0 to be able to change the MSBs and LSBs of a frequency word separately. When DB13 (B28) = 1, this control bit is ignored. HLB = 1 allows a write to the 14 MSBs of the addressed frequency register. HLB = 0 allows a write to the 14 LSBs of the addressed frequency register.
DB11	FSEL	The FSEL bit defines whether the FREQ0 register or the FREQ1 register is used in the phase accumulator. See Table 8 to select a frequency register.
DB10	PSEL	The PSEL bit defines whether the PHASE0 register data or the PHASE1 register data is added to the output of the phase accumulator. See Table 9 to select a phase register.
DB9	PIN/SW	Functions that select frequency and phase registers, reset internal registers, and power down the DAC can be implemented using either software or hardware. PIN/SW selects the source of control for these functions. PIN/SW = 1 implies that the functions are being controlled using the appropriate control pins. PIN/SW = 0 implies that the functions are being controlled using the appropriate control bits.
DB8	RESET	RESET = 1 resets internal registers to 0, this corresponds to an analog output of midscale. RESET = 0 disables RESET. This function is explained in the RESET Function section.
DB7	SLEEP1	SLEEP1 = 1, the internal MCLK is disabled. The DAC output remains at its present value as the NCO is no longer accumulating. SLEEP1 = 0, MCLK is enabled. This function is explained in the SLEEP Function section.
DB6	SLEEP12	SLEEP12 = 1 powers down the on-chip DAC. This is useful when the AD9834 is used to output the MSB of the DAC data. SLEEP12 = 0 implies that the DAC is active. This function is explained in the SLEEP Function section.

Bit	Name	Description
DB5	OPBITEN	The function of this bit is to control whether there is an output at the SIGN BIT OUT pin. This bit should remain at 0 if the user is not using the SIGN BIT OUT pin. OPBITEN = 1 enables the SIGN BIT OUT pin. OPBITEN = 0, the SIGN BIT OUT output buffer is put into a high impedance state, therefore no output is available at the SIGN BIT OUT pin.
DB4	SIGN/PIB	The function of this bit is to control what is output at the SIGN BIT OUT pin. SIGN/PIB = 1, the on-board comparator is connected to SIGN BIT OUT. After filtering the sinusoidal output from the DAC, the waveform can be applied to the comparator to generate a square waveform. Refer to Table 17. SIGN/PIB = 0, the MSB (or MSB/2) of the DAC data is connected to the SIGN BIT OUT pin. Bit DIV2 controls whether it is the MSB or MSB/2 that is output.
DB3	DIV2	DIV2 is used in association with SIGN/PIB and OPBITEN. Refer to Table 17. DIV2 = 1, the digital output is passed directly to the SIGN BIT OUT pin. DIV2 = 0, the digital output/2 is passed directly to the SIGN BIT OUT pin.
DB2	Reserved	This bit must always be set to 0.
DB1	MODE	The function of this bit is to control what is output at the IOUT pin/IOUTB pin. This bit should be set to 0 if the Control Bit OPBITEN = 1. MODE = 1, the SIN ROM is bypassed, resulting in a triangle output from the DAC. MODE = 0, the SIN ROM is used to convert the phase information into amplitude information, resulting in a sinusoidal signal at the output. See Table 18.
DB0	Reserved	This bit must always be set to 0.

A2: Software

Overall System Software Flow Chart

The first flow chart shows the basic steps of the overall system. Before the interrupt is enabling, PORTC for keypad, PORTB for LCD and Timer1/Counter1 are configured and initialized. Then the system just waits for the interrupt. When the interrupt occurs, the system will find out what key is pressed and pass the key to the finite state machine, which is explained in section "FSM".


Initialization
Configure PORTB for LCD in SPI interface
Configure PortC for keypad. PD0 as interrupt pin
Initialize Timer1/ Counter1
Enable interrupt
Initialize the LCD
No interrupt
Wait for interrupt
Interrupt occurs
Find out the keyvalue
Call FSM