International Journal of Engineering Research and General Science Volume 4, Issue 4, July-August, 2016
ISSN 2091-2730

Low Cost Controller Platform for Power Electronic Application

Aditya Porwal, Karan Varshney

IIT Delhi

aporwal25@gmail.com

Abstract— A DSP (Digital Signal Processor) is a basic controller platform which can be interfaced between a computer and a machine. It can be programmed to control the machines in a desired way for various power electronic applications. This project is based on an idea of replacing a typical DSP by a low cost alternative. As an alternative, it would consist of a micro-controller working as a processing unit. Other peripherals like Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC) can be interfaced with the micro-controller depending upon the compatibility of the micro-controller. The report encapsulates the results of controlling a 3-phase induction motor using the Sinusoidal Pulse Width Modulation (SPWM) module developed, using a pre-programmed Voltage per Hertz (V/f) control method[1]. In the end comparison between a DSP and a micro-controller is included.

Keywords—digital signal processor, V/f control, controller platform, pulse width modulation, microcontroller, Arduino controller

INTRODUCTION

A DSP is a specialized micro-processing unit programmed according to the operational needs of digital signal processing. A basic DSP block is interfaced through its ADC and DAC. Analog signal is received by ADC which is sent to the processing unit of the DSP in digital domain. For power electronic applications, the goal of DSP is to follow the specific algorithms for controlling the machine parameters or the output signal[3]. DSP can either transmit the processed signal in digital format or in analog format with the help of DAC.

For most of the power electronic applications, a DSP can be replaced by a general purpose micro-processors like Arduino or Raspberry Pi. These micro-processors can be programmed as per the needs of application. Programming these devices is flexible as they can make use of third-party libraries and other modules on a higher machine level. This allows it to be used as a processing unit in several applications. Hence, the same micro-controller board can be interchangeably used for different kinds of power electronic applications by simply making use of different predefined programs. The merits of replacing a DSP with a generic micro-controller includes its low cost, compact size and high processing speed.

In the performed experiment, the feasibility and the extent of replacing a DSP with a micro-controller is established. The limitations of such a replacement and the fields where a DSP can be interchangeably used with a micro-controller is studied. As a part of the experiment, the micro-controller is used to run a 3-phase induction machine with an open loop control system. However, for implementing a closed loop control system a feedback to the micro-controller is required, this can be achieved by interfacing an ADC to the micro-controller. One of the key feature of the experiment is to introduce dead time through the control program rather than using hardware to introduce dead time. With a control on the dead time, the safety of the driving system and the overall control system can be ensured.

IMPLEMENTATION

Sinusoidal Pulse Width Modulation

In Sinusoidal Pulse Width Modulation, a sine wave of frequency f_s is sampled by a triangular wave of frequency f_t. During a period of the triangular wave, the sinusoidal signal is sampled and approximated by a constant DC signal of value V_{dc}. In this modulation scheme, the pulse width is varied in accordance with the sampled value of sinusoidal signal V_{dc}.

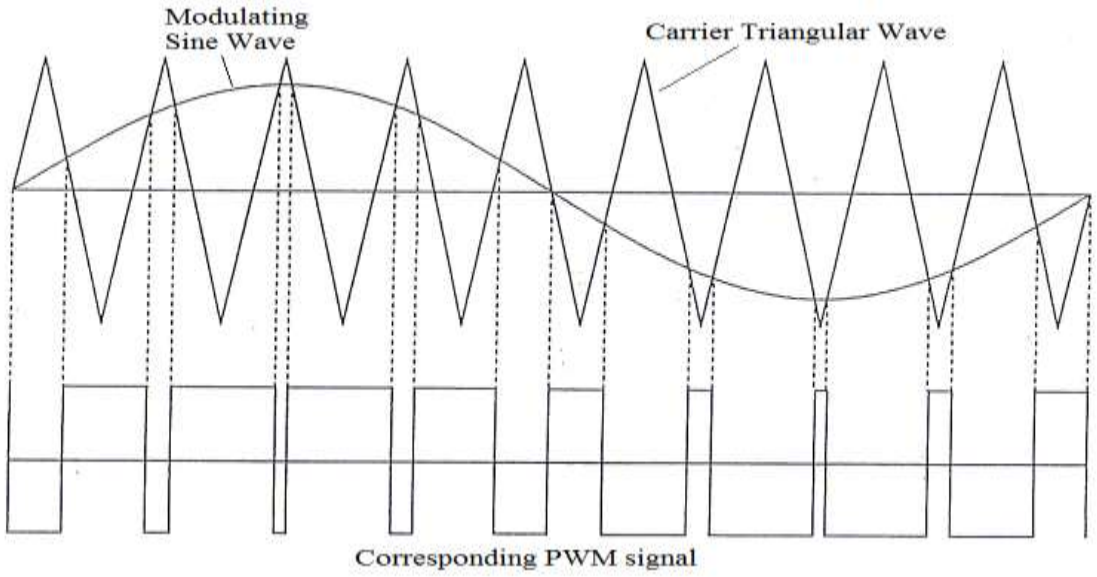


Figure 1. Sine PWM generation

Hardware and Software PWM

For implementing SPWM, the reference sinusoidal signal can be fed to the micro-controller through an ADC as a feedback for closed loop control or else the micro-controller can be pre-programmed to use tabulated samples of the sinusoidal wave. Many micro-controllers provide a hardware interface for PWM. Generally 8-bit or 16-bit counters are present, which are compared against a set of compare registers every clock cycle. Whenever the values in the counter and the compare register matche, the output is changed according to the specified mode. Various kinds of modes provided by a micro-controller are phase correct mode, frequency correct mode and phase-frequency correct mode. For triangular wave, the counters need to be up-down counter which corresponds to phase correct mode. Thus by controlling the value in the compare register, corresponding to V_{dc}, desired pulse width can be achieved.

Another way to implement this would be to use any GPIO pin to toggle ON/OFF based on the control program written. In this method, known as software PWM, the output is generated by comparing the value of the V_{dc} against a counter which gets updated after every fixed number of clock cycles. Whenver the value matches the output is modified by the control program. To generate error free output, the counter must be updated very fast,i.e., almost every cock cycle. But in practice, when the program runs on the controller the counter gets updated after hundreds-thousands clock cycles, this happens because the processor needs to go through the overall control loop before updating the counter. Clearly software PWM requires large execution time and hence it is slow[2]. In generality, software PWM is good when the sampling frequency is low, as high frequencies cannot be achieved due to the inherent reasons[12].

Benefits of hardware PWM over software PWM includes the ability to generate more than one PWM channels simultaneously, processor is free to process other information without interrupting PWM channels, precise sampling frequencies can be generated.

Dead Time Incorporation

To drive a 3-phase 2-level invertor, the gate signals to the top and the bottom gates must be complementary. In case if both the top and the bottom gate signals are high then the power supply gets shorted, to avoid accidental shortening of the power supply, a dead time needs to be introduced in the two signals. This is like a safety margin to take care of the turn-on and turn-off time of the switches. In practice driving circuitry introduces rising delay and falling delay in the two driving signal which are supposed to be complementary of each other. Such delays can make the two signals high for a short interval of time. To introduce a dead time (dead band) in a signal either its rising edge is delayed or its falling edge is advanced by a fixed amount of time[6].

The need of having dead time between such signals is clear, especially when the power supply is in hundreds of volts. There are different ways to introduce dead time between such signals. One method employs the use of dedicated circuitry to introduce dead time. Although, in this experiment the dead time would be introduced with the help of the control program i.e., through software. Such method would allow a better control on the dead time to ensure the safety of the system. Apart from this, it will not require extra circuitry and hence reduce the size of the control circuitry.

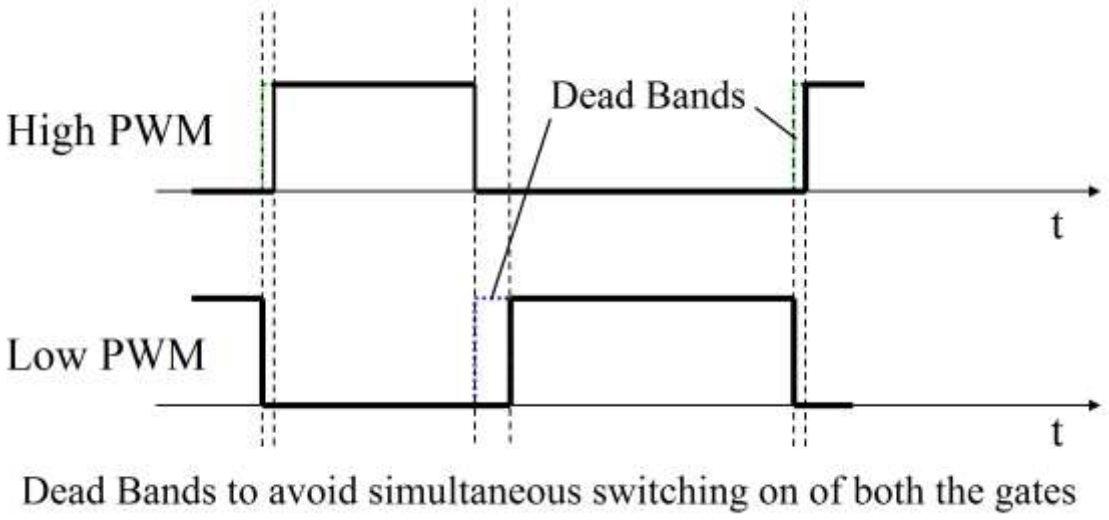


Figure 2. Complementary Signals with dead time in them

In this project, two low-cost microcontrollers are considered for the SPWM implementation, Arduino and Raspberry PI.

Working with Arduino

Arduino Mega2560 peripheral includes 12 PWM channels with programmable resolution from 2 to 16 bits along with four 16bit timer/counter with separate prescaler and compare mode. On varying the prescaler, the time to reach peak and fall down (or in turn frequency) can be varied and that's how hardware PWM can be modified according to the need[4]. The peripheral also provides 16 channels of 10bit ADC and no external interfacing is required for ADC in closed loop implementation. 86 programmable input/output lines (General Purpose Input Output) are availaible (including PWM and ADC channels) in peripheral and any GPIO pin of our choice could act as output.

PWM Implementation

In case of software PWM, the counter used for comparison against the sampled value of the sine wave, cannot be synchronized properly with the system clock. The code execution requires time for evaluation and it depends upon the availability of the processor. As long as the sampling frequency is small it would work but its performance would start deteriorating as the frequency is increased. Also software PWM is slow for the reasons discussed in the last section. On the other hand hardware completely devoted to PWM is

already in synchronization with the system clock and the sampling frequency can be controlled by setting the prescaler value and the top value in the PWM registers appropriately. Arduino provides 16-bit PWM channels along with 8-bit channels which provides better precision in terms of setting the sampling frequency.

PWM Registers

For using hardware PWM channels the registers associated with it must be initialized properly. A 16-bit timer in Arduino has three main control registers to control the channel operation, namely, *TCCRxA*, *TCCRxB* and *TCCRxC*.

Bit	7	6	5	4	3	2	1	0	
(0x80)	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	

Bit	7	6	5	4	3	2	1	0	
(0x81)	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	

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Figure 3. TCCRxA, *TCCRxB* and *TCCRxC* registers format

Apart from them it has one 16-bit counter register, namely, *TCNTx* for keeping track of counter value and three output compare registers for setting the reference values, namely, *OCRxA*, *OCRxB* and *OCRxC* to provide three different kind of outputs per channel.

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
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	

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
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 4. OCRxX and *TCNTx* registers format

A register Input Capture Register, *ICRx* for updating the counter value whenever an event occurs on input capture pin of the channel. Particularly useful in closed loop implementation.

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
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 5. ICRx register format

For handling interrupts, two registers are provided: one *TIMSKx* for enabling which interrupt vectors are to be used and another *TIFRx* which gets updated as soon as interrupt is generated and handled.

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 6. TIMSKx and TIFRx register format

Interrupts

Only single instruction set at a time can be executed by a processing unit, but instructions can be interrupted. Therefore, for multitasking, the events are prioritized for the processor to process in a queue. Hence in the given preference order the interrupts are handled by the interrupt handler (or ISR, Interrupt Service Routine)[7]. As soon as the interrupt is flagged for an instruction, current instruction is paused and is resumed once the flagged instruction is over.

Interrupts are required to set the new compare/reference value for each of the PWM channels based on the sine wave, every period of the triangular wave. To introduce dead time, it needs to be updated twice every period of triangular wave. This enables the processor to be free to process other data such as in the case of closed loop system, the feedback needs to be processed which in turn can be used in the next sampling cycle.

Memory

Arduino Mega2560 has only 256 KB of FLASH memory which can only be written at the time of flashing the micro-controller and SRAM is limited to 8 KB only. Hence in case of storing large data such as parameters, look-up tables cannot be stored in SRAM, one needs to use the FLASH memory. This limits the smooth operation of the system.

Clock speed

The system clock speed of the Arduino Mega2560 is 16 MHz and correspondingly the maximum MIPS it could achieve at 16 MHz is 16 MIPS. Whereas the clock speed for typical DSPs is 150-200 MHz which insures a much higher MIPS than Arduino. Hence, the amount of computation that can be done in each cycle of the sampling wave is limited to very few operations. In closed loop implementation, this plays an important role since the new reference value has to be calculated from the feedback which can involve complex arithmetic operations which needs to be finished before the next cycle of the sampling wave.

Working with Raspberry PI

PWM Implementation

Similar to Arduino, software PWM faces the same problem of synchronization when the sampling frequency is high. Raspberry PI provides only 2 hardware PWM channels with only a single output per channel. Incapability to generate 3 phase signals on the same micro-controller is the major reason which makes the use Raspberry PI in 3 phase power electronics application difficult.

PWM Registers

The hardware PWM channels of Raspberry PI provides very less flexibility. The counter available is down counter only in contrast to Arduino which has down counter, up counter and up-down counter modes. Up-down counters are required for triangular sampling

wave. The sampling frequency can only be controlled by changing the top value of the counter, which corresponds to the maximum value of the sampling wave but in Arduino even the width of the sampling wave can be controlled.

Absence of ADC on the micro-controller requires to interface an external ADC for closed loop implementation.

Interrupts

No predefined interrupt structure is available for the PWM hardware channels. Even user defined interrupts will not work at such high frequencies. The main reason being that Raspberry PI runs a Linux based operating system which is not a real time operating system. Thus program that runs on such a platform does not get updated in real time, this causes problem because when interrupt is fired it is not handled immediately. The same reason makes software PWM difficult because the parameters cannot be updated at fixed regular intervals.

Memory and Kernel Drivers

Raspberry PI provides 512 MB RAM significantly higher than a DSP and Arduino and extendable external storage. It allows to store a large amount of preprocessed data which can be used by the program as and when needed.

The problem of Linux being not a real time operating system can be bypassed by flashing a program developed for carrying out the only required function. This can be achieved by making use of kernel drivers and developing the code in assembly language making the use of the Raspberry PI's *SoC BCM2835* registers and then flashing the program on the micro-controller similarly as in the case of Arduino.

Clock Speed

Raspberry PI B+ was used for experimentation. It has significantly high clock speed (~700 MHz) as compared to a regular DSP (~200 MHz). Even the chip can be overclocked externally. Such high operating frequency allows it to do more computation with faster processing speed than a typical DSP. Big complex feedback algorithms can be implemented easily in case of closed loop implementation.

Experimentation

As per the points mentioned above, although Raspberry PI has two main features of large memory and high clock speed but lack of hardware PWM channels and inability to do software PWM at high sampling frequencies makes Raspberry PI less of a substitute for a DSP. Whereas Arduino has a wide flexibility in terms of PWM and the only problem is its low clock speed. Arduino's memory limitation can be avoided by using external memory modules which are readily available. Hence replacing a DSP with Arduino will work just as fine if open loop implementation of the system is carried out. In closed loop implementation, it is customary for the DSP to do several computations based on the feedback and then correspondingly update the system parameters. Owing to the low clock speed the amount of computation that can be done with Arduino is limited. Apart from that in terms of PWM generation Arduino can successfully replace a DSP.

For the current experimentation, Arduino is used rather than Raspberry PI. This experiment will illustrate the open-loop control of a three phase induction machine via Voltage per Hertz control method. The controlling unit will be Arduino which will generate the six gate driving signals based on the parameters provided which will drive the three phase inverter and hence the induction machine.

Interface Circuitry

(For Interface Circuitry refer Appendix)

The Arduino generates the 6 gate driving signals which are being fed to the 3 phase two level inverter. This requires an interfacing circuitry as Arduino cannot drive the inverter directly, due to the limited current carrying capability of Arduino, which is less than 10 µA. Thus a buffer is introduced in between and an optocoupler to provide electrical isolation between the two circuits for safety.

Arduino Initialization

(For Codes refer Appendix)

The INO file to be flashed into Arduino is generated by a C++ program which takes the input for the operating conditions. First it calculates the appropriate parameter values and the needed sine look-up table. After this it generates the INO file which than can be flashed. This is required since the sine look-up is large and need to be stored in the Program Memory, i.e., Flash memory of Arduino. Preprocessing the sine table reduces the time taken to compute the next reference value and thus leaves more time for other computation during the sampling period such as computation on feedback.

To setup the hardware channels of the Arduino the required 16-bit timer's control registers need to be updated. To get a triangular wave, i.e., an up-down counter the timer's mode needs to be set to *Phase Correct* PWM mode such that their top value can be controlled through code. This requires the *WGMx [3:0] = 1011* in the control registers of the timer. For each of the three timers, three different kind of output can be generated, which can be controlled by the values of *COMxA [1:0]*, *COMxB [1:0]* and *COMxC [1:0]*. It is required that one output gets clear when counter matches the reference value while up-counting and set while down-counting and the other output to do the vice versa and the last output is not required.

Hence, *TCCRxA = 00111011* and *TCCRxB = 00010000*.

Also, to get the desired sampling frequency the top value of the counter and the timer's clock can be prescaled by a factor of powers of 2. The appropriate values of the prescaler and the top are found based on the desired frequency by a simple algorithm.

Hence, the value of the prescaler is set in the *TCCRxB* and top value is stored in the *OCRxA.*

Now the reference values are to be set in the *OCRxB* and *OCRxC* registers which needs to update whenever the counter reaches top or bottom. This is required because the dead time is introduced by pre-processing the values. To achieve this interrupts from one of the timer are generated whenever such an event happens. To set this any one of the three timer's *TOIEx* and *OCIExA* bits in *TIMSKx* are set to one and all the other timers to zero.

Hence, for any one timer *TIMSKx = 00000011* and for other timers *TIMSKx = 00000000.*

Thus whenever the interrupt occurs the reference values, i.e., *OCRxB* and *OCRxC* updates based upon the next value in the look-up sine table which is generated beforehand the Arduino is flashed with the program. It is also needed in Voltage per Hertz control the frequency needs to be ramped from some starting value to the final value proportional to the sine amplitude. For this the reference values of that sine wave is processed and stored in memory.

Thus whenever the program runs in Arduino, the reference value gets updated by looking in the table corresponding to that frequency at that particular time of ramping. Therefore, the C++ program is required to generate a series of sine look-up table at different frequencies of ramping. The Arduino program also keep track of the ramping time and the ramping is stopped one the desired frequency is reached.

Execution Order:

Parameters → C++ Code → (If Possible) → INO File → Arduino Flash →Run → Observations

The parameters are passed to a C++ code and it checks if it is possible to implement the given configuration on Arduino. If it is possible this code generates a INO file which can be flashed on Arduino and then the Arduino can be used accordingly.

Results

The results are generated for sampling frequency of 5 KHz and fundamental frequency of 50 Hz with a 3 µs dead time for a balanced 3 phase.

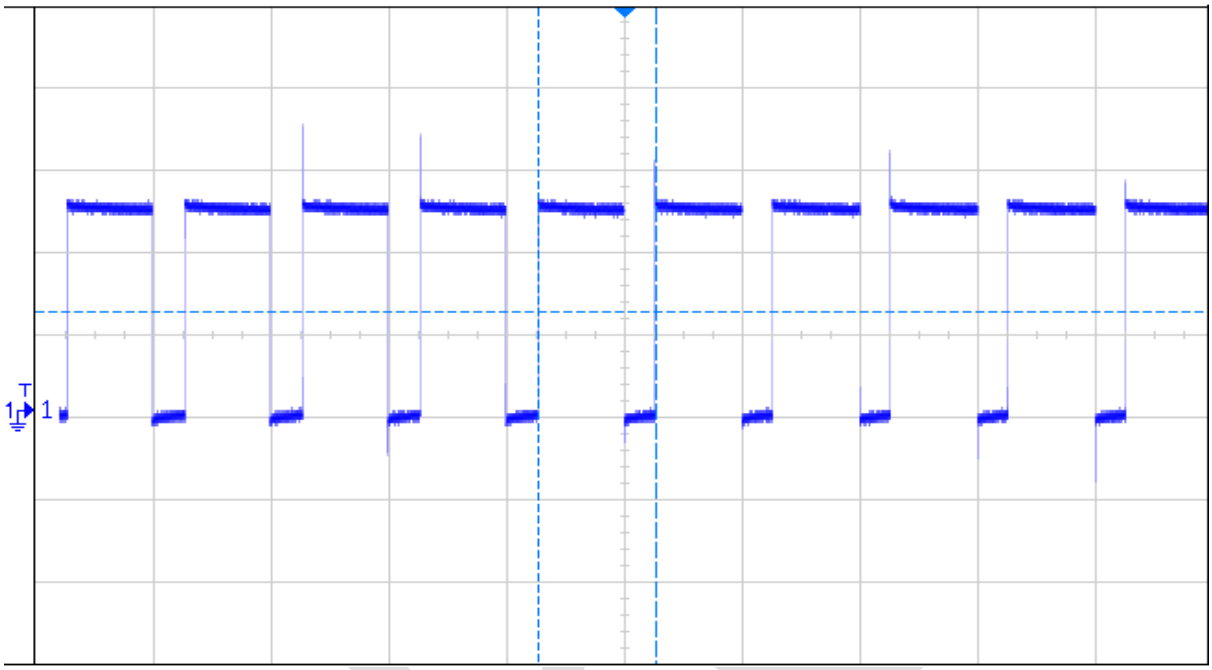


Figure 7. Sine PWM Output of a channel (Time Scale 200 µs)

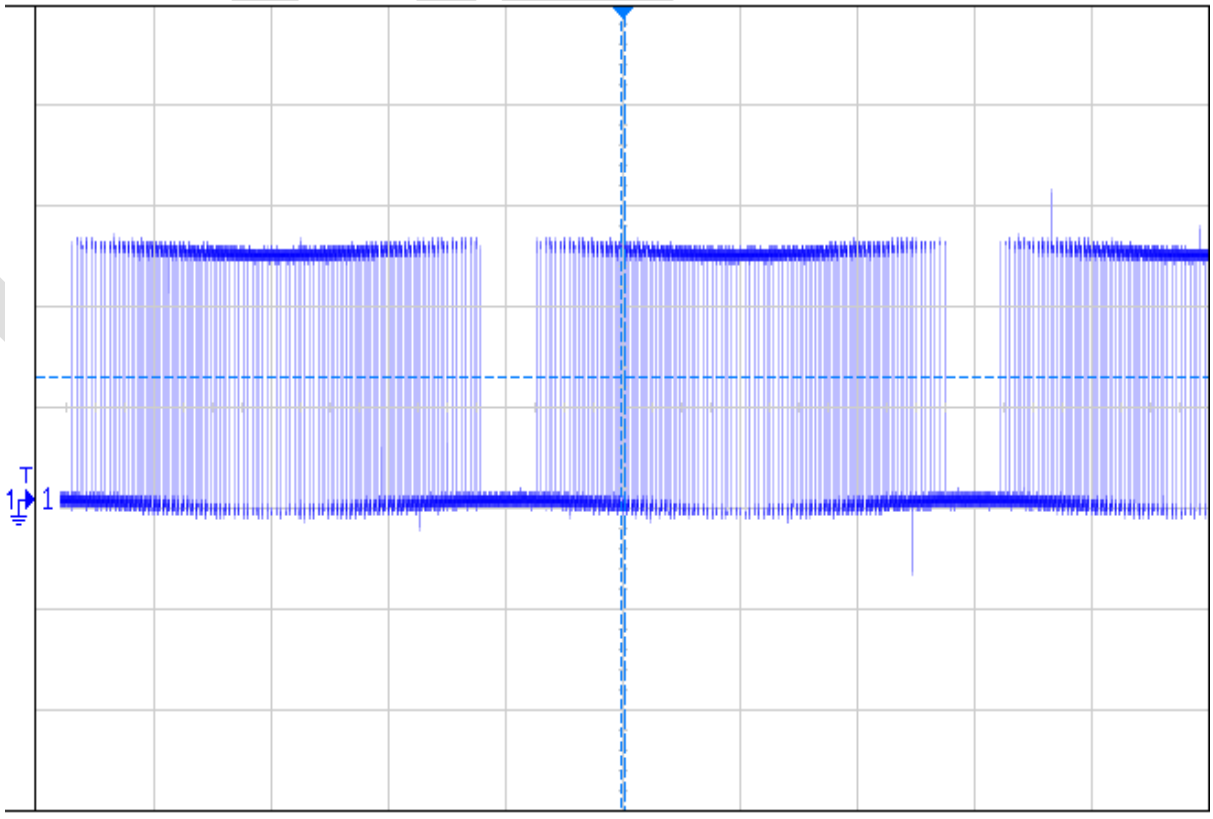


Figure 8. Sine PWM Output of a Channel (Time Scale 5 ms)

Figure 7 and 8 depict the output voltage of phase. The switching time of 200 µs corresponding to the 5 KHz sampling frequency and at large time scale the fundamental period of about 20 ms can be observed corresponding to the fundamental frequency of 50 Hz.

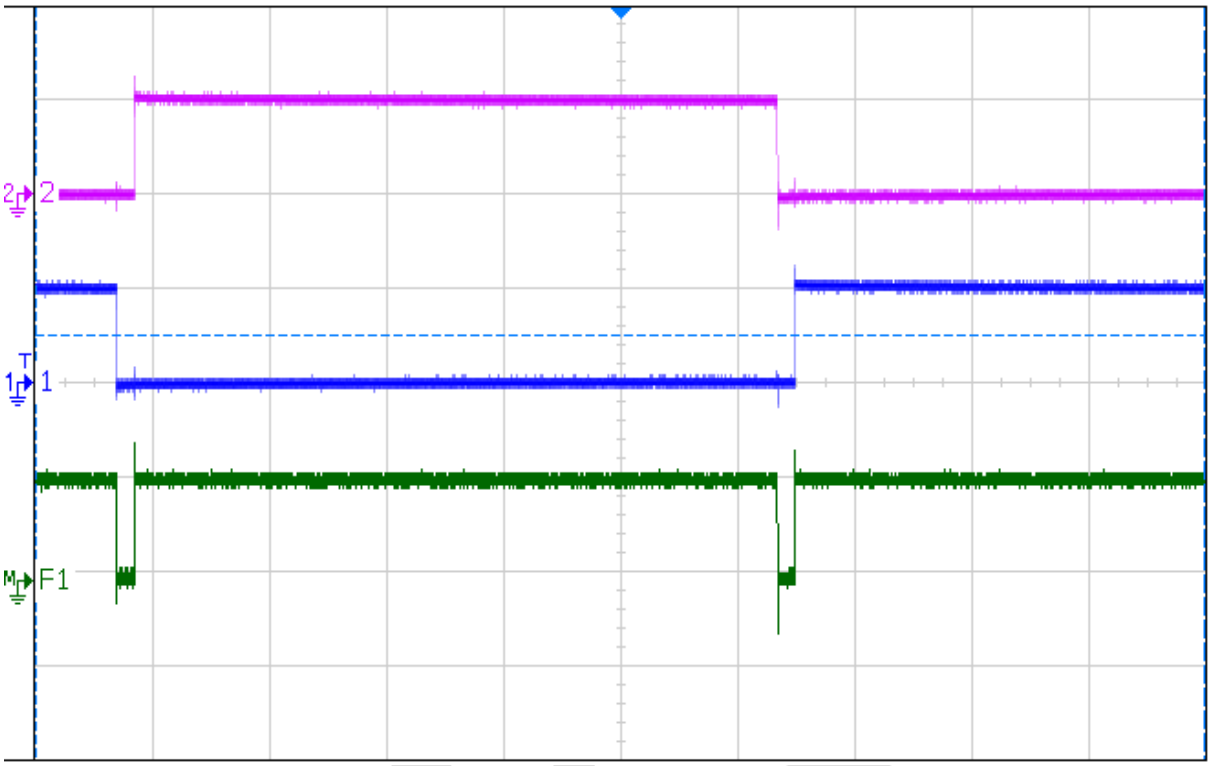


Figure 9. Dead Time (Channel M = Channel 1 + Channel 2) (Timescale 20 µs)

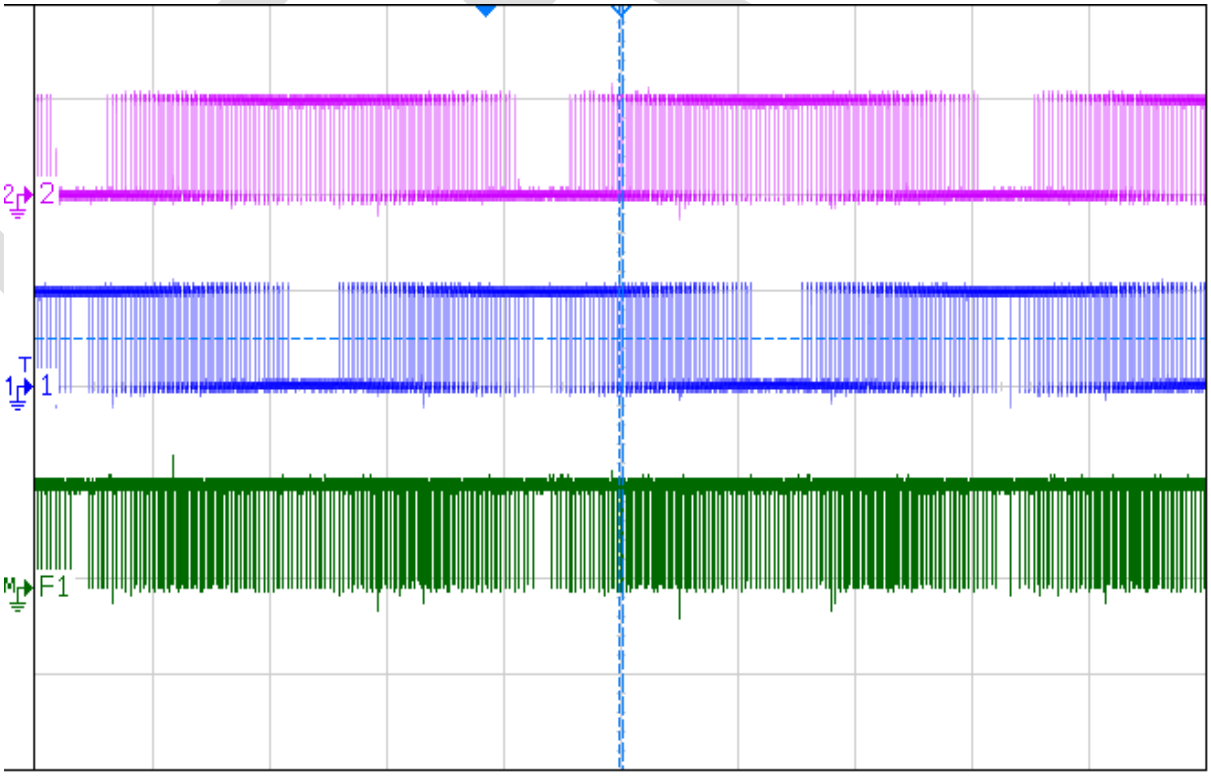


Figure 10. Dead Time (Channel M = Channel 1 + Channel 2) (Timescale 5 ms)

In figure 9 and 10, channel 1 and channel 2 are complementary signals which are fed to one of the 3 pairs of the top-bottom IGBTs. The signal *M* is the sum of channel 1 and channel 2 and all the three signals are scaled to 5 V/block, thus *M* represents the dead time of 3 µs each, between the two channels. Channel *M* never goes beyond 5V implying that channels *1* and *2* are never high at the same time.

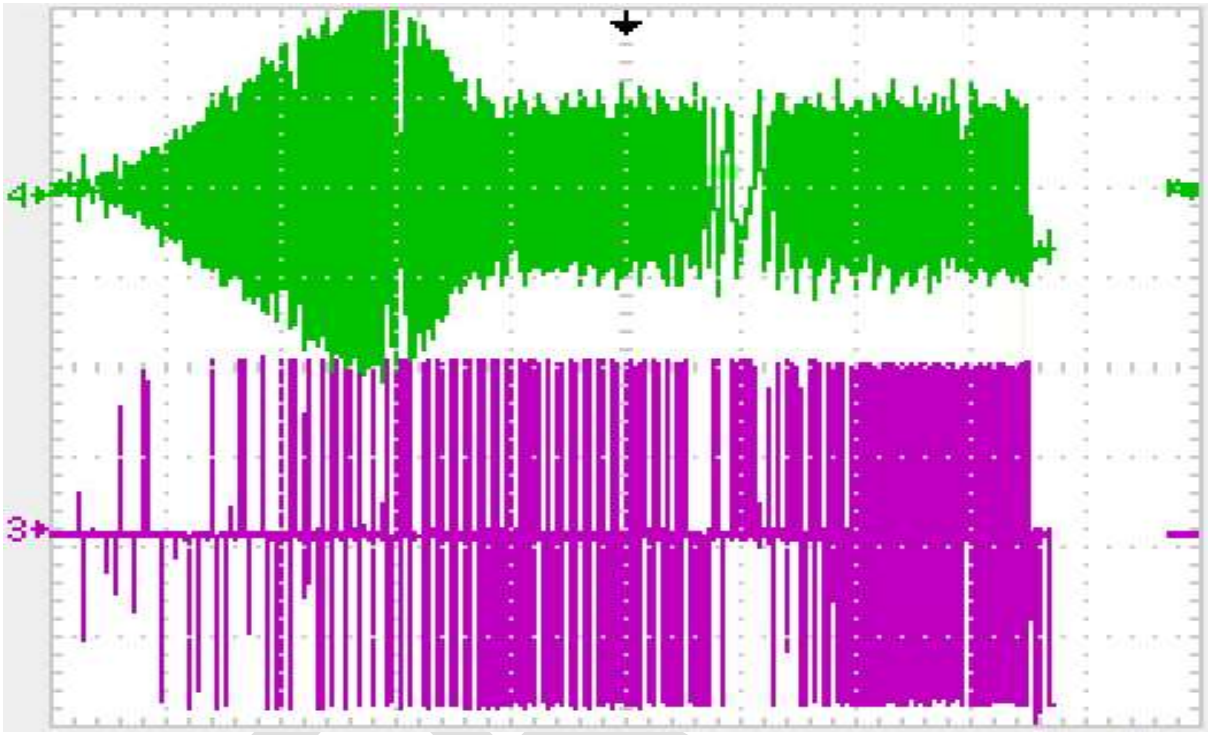


Figure 11. Channel 3 is voltage across one of the phase of 3 phase induction motor and Channel 4 is the current profile flowing in that phase during ramping (V/f control).

Figure 11, shows voltage and current across a phase when the *V/f* control is done. The machine is started at very low frequencies compared to the fundamental frequencies. At regular interval the frequency is increased in small steps keeping the flux almost same. Initially the voltage and the frequency is small and the motor is at standstill, slowly the current in the motor builds and the slip of the motor starts decreasing and as the motor goes into steady state operation the current amplitude becomes constant. During the transient the current rises sharply and overshoots the steady value due to the inertia of the motor which leads to high slip. This can be reduced by increasing the ramping time.

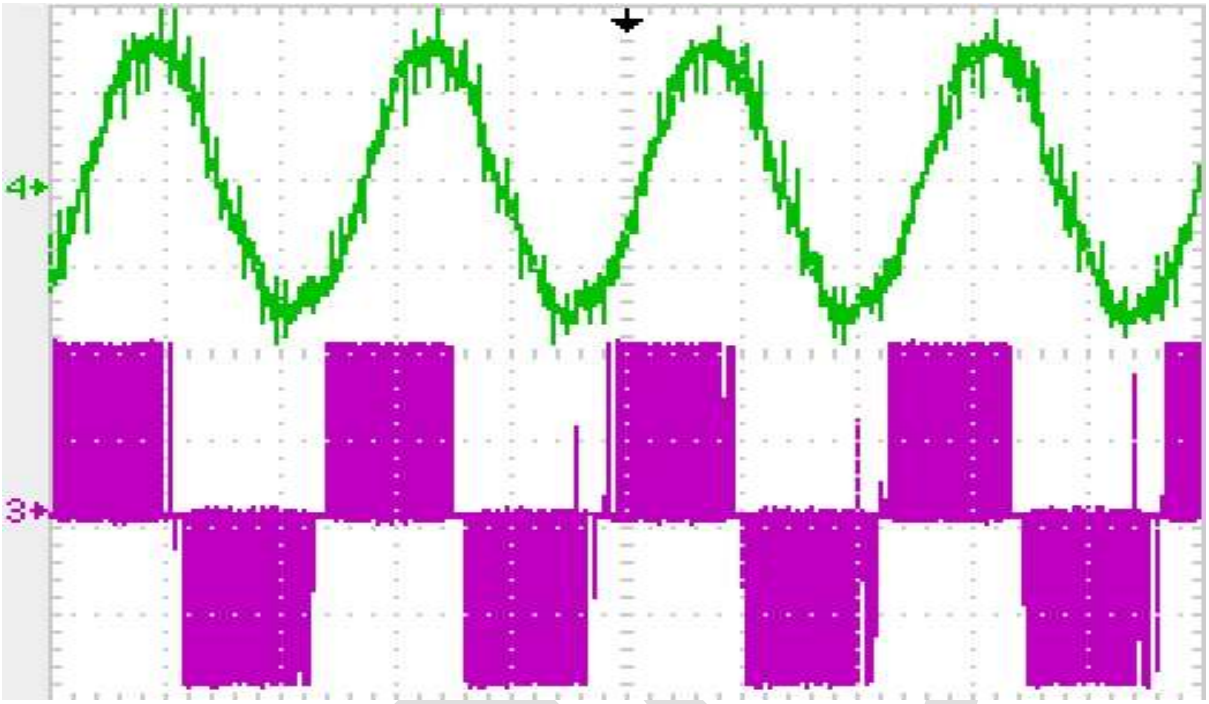


Figure 12. Channel 3 is voltage across one of the phase of 3 phase induction motor and Channel 4 is the current profile flowing in that phase.

Figure 12, shows the voltage across a phase of the 3 phase induction motor and current through that phase which is fairly a sinusoidal signal of 50 Hz. This voltage is simply the voltage available across one of the phase of the inverter which is nothing but the difference of the two signals generated by Arduino and amplified to the power supply level. Sinusoidal nature of the current is due to the large inductance of the motor allowing only smaller frequencies and filtering out higher harmonics which are present in the voltage waveform applied across it. Fourier analysis of this voltage signal reveals 50 Hz component and its harmonics are present in the signal.

CONCLUSION

Two methods of PWM, namely, software and hardware are available but software PWM is inefficient at high sampling frequencies, which is generally the case in power electronics application. Even at sampling frequencies as low as 1 khz, the software PWM output is not steady and accurate as it depends upon the availability of the processor. Thus hardware PWM is the only feasible option for PWM at such frequencies, where the processor is needed to only do the system parameters update and feedback computation.

Generally, dead time is introduced with the help of circuitry but in the experiment the dead time is generated as a part of the pulse width modulation scheme. This helps in getting rid of the circuitry needed to introduce the dead time in the signals.

Based on the results of the above experiment, Arduino can successfully replace a DSP for any pulse width modulation scheme. In open loop implementation Arduino can safely replace a DSP but in closed loop implementation the amount of computation on the feedback is limited. The inbuilt ADC of Arduino has a 10-bit precision which can be improved by interfacing a better ADC.

The Raspberry PI can prove to be a better substitute for a DSP because it has significantly high clock speed and large memory if only Raspberry PI have a better hardware PWM module. Even though making use of kernel drivers one can achieve software PWM but the performance is limited in terms of the sampling frequency. For now the current versions of Raspberry PI are incapable of replacing a DSP for a typical PWM application.

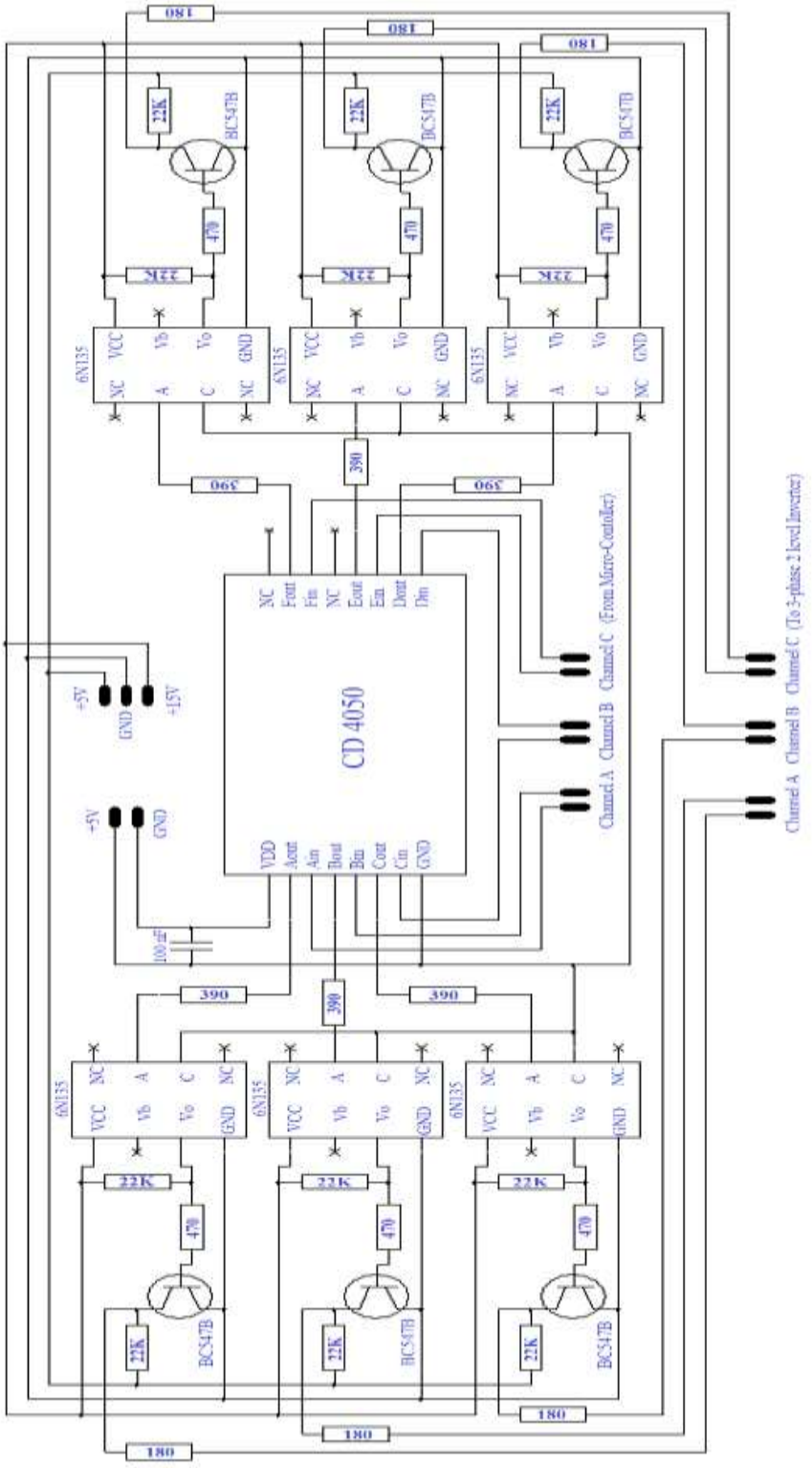
Such micro-controllers such as Arduino can easily substitute a DSP as a low cost alternative. This enables various individuals, firms, laboratories and students to use a micro-controller in place of a DSP for a variety of power electronic applications and experimentation purposes in schools and colleges.

REFERENCES:

[1] A. Munoz-Garcia "A new induction motor V/f control method capable of high-performance regulation at low speeds" IEEE Transactions on Industry Applications, ISSN-number 0093-9994, Jul/Aug 1998
[2] J. P. Agrawal , O. Farook ; C. R. Sekhar "Software controller for PWM converters" Applied Power Electronics Conference and Exposition, 1994. APEC '94. Conference Proceedings 1994., Ninth Annual, ISBN-number 0-7803-1456-5, 13-17 Feb 1994
[3] A. H. M. Yatim; N. R. N. Idris "Implementation of direct torque control of induction machine utilising TMS320C31 digital signal processor" Signal Processing and its Applications, Sixth International, Symposium on. 2001, ISBN-number 0-7803-6703-0, 2001
[4] Milan Matijevic; Vladimir Cvjetkovic "Overview of architectures with Arduino boards as building blocks for data acquisition and control systems" 2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV), ISBN-number 0-7803-6703-0, 24-26 Feb. 2016
[5] Piotr Biczel; Andrzej Jasinski; Jacek Lachecki "Power Electronic Devices in Modern Power Systems" EUROCON, 2007. The International Conference on "Computer as a Tool", ISBN-number 978-1-4244-0813-9, 9-12 Sept. 2007
[6] L. Schirone; M. Macellari "General purpose dynamic dead time generator" Clean Electrical Power (ICCEP), 2015 International Conference, ISBN-number 978-1-4244-0813-9, 16-18 June 2015
[7] Yuting Zhang; Richard West "Process-Aware Interrupt Scheduling and Accounting" 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06), ISBN-number 0-7695-2761-2, Dec. 2006
[8] V. Miñambres-Marcos; I. Roasto; P. Szczepankowski; E. Romero-Cadaval; D. Vinnikov; F. Barrero-González "Code development of a DSP-FPGA based control platform for power electronics applications" 2015 IEEE International Conference on Industrial Technology (ICIT), 17-19 March 2015
[9] M. F. Rahman; K. V. Baburaj; L. Zhong "A DSP laboratory platform for teaching power electronics and drives" 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06), ISBN-number 0-7695-2761-2, Dec. 2006
[10] A. Balestrino; G. de Maria; L. Sciavicco "Analysis of modulation processes in power convertors" Electrical Engineers, Proceedings of the Institution of, ISSN-number 0020-3270, May 1978
[11] Babak Zamanlooy; Hamidreza Chamani Takaldani; Amir Moosavienia; Khalil Monfaredi; Reza Ebrahimi Atani "Design and implementation of a stable platform digital controller based on DSP" Signals and Electronic Systems, 2008. ICSES '08. International Conference on, ISBN-number 978-83-88309-47-2, 14-17 Sept. 2008
[12] M. C. Trigg; H. Dehbonei; C. V. Nayar "Digital sinusoidal PWM generation using a low-cost micro-controller based single-phase inverter" 2005 IEEE Conference on Emerging Technologies and Factory Automation, ISBN-number 0-7803-9401-1, 19-22 Sept. 2005

APPENDIX

A. The schematic of the interface circuitry is as below:



The 3-phase from the micro-controller are fed to the buffer to prevent excessive current flowing into the microcontroller. These signals are passed through an optocoupler stage to provide an electrical insulation between the two stages. These 5V signals are then amplified to 15V through a transistor, which is needed to drive the gates of an inverter.

The three output channels are connected to the three legs of the inverter for driving the 3 phase induction machine by converting the input

AC or DC link voltage to the inverter.

B. C++ program code for generating the INO file can be found here:
https://docs.google.com/document/d/1HN1Oc3OyGnYVTc5W-0tXclh6qCX1ANrNDV6bf_xqyKc/edit?usp=sharing

C. Arduino program generated by the above code for 10 KHz sampling frequency, 50 Hz fundamental frequency, dead time of 3 µs, ramping time of 60 seconds with a frequency step of 1 Hz starting from 5 Hz to produce a 3 phase balanced output can be found here:
https://docs.google.com/document/d/1GgGYg2ZBTsnUOeruSHhgLMpD3RzctSSuCKd75SLIQq0/edit?usp=sharing