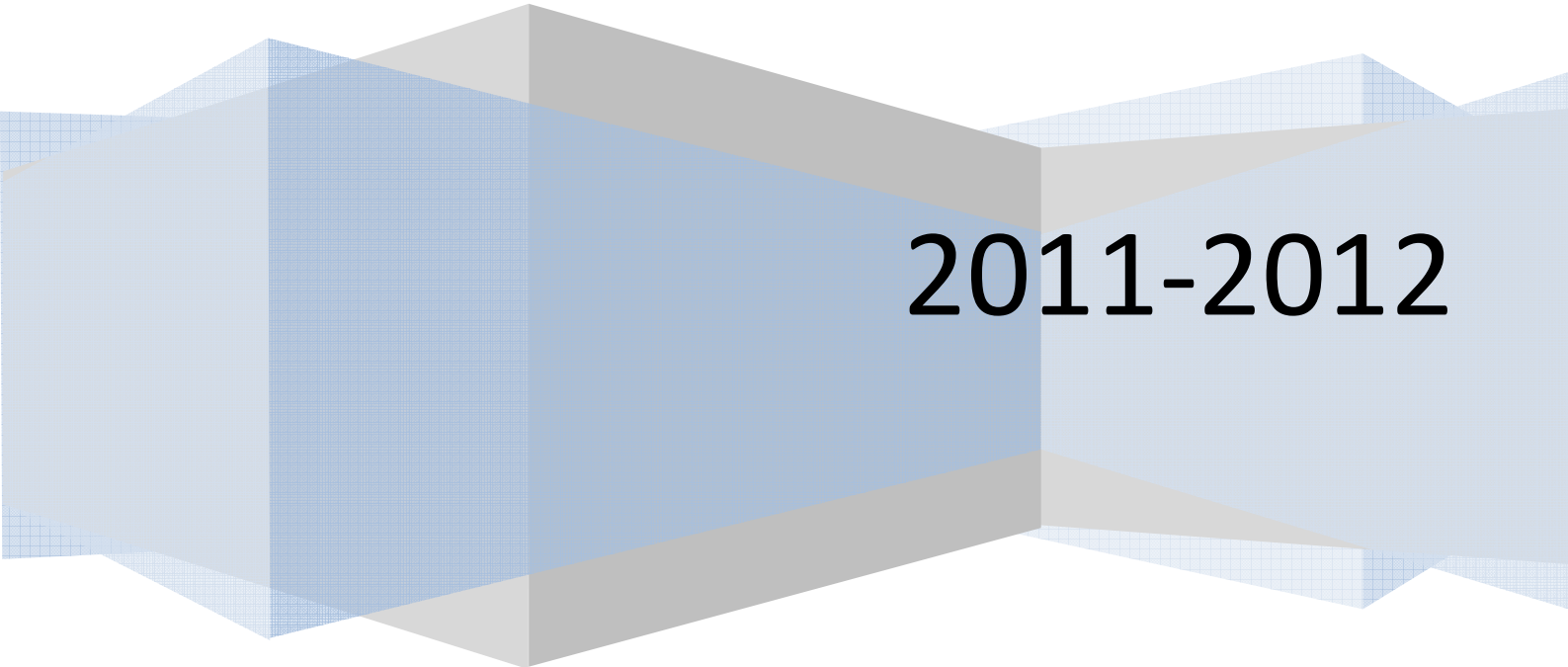


Ecole Nationale d'ingénieurs de Tarbes - France

European Project Semester “Multilevel Inverter”

EPS

Javier Pérez Germán



2011-2012

INDEX

1. INTRODUCTION	Page numbers:
1.1 Primes	3
1.2 Original assignment from Enit	4
1.3 Clients and requirements	6
1.4 Scope	6
2. POWER MODULE	
2.1 Research information	7
2.2 State of art	9
2.3 Choose the structure	13
2.4 Choose the components	16
2.5 Simulation circuit in PSIM software	18
2.6 Design the circuit to print circuit board	19
2.7 Ensemble the power components	21
2.8 Power Tests	22
3. CONTROL MODULE	
3.1 Research information	23
3.2 Accomodate values	26
3.3 Chose components	27
3.4 Acquire software	27
3.5 Designing and testing the C Software	28
4. COOLING MODULE	
4.1. Why do we need a cooling system?	34
4.2. Finding the right components	35
4.3. Calculations of the Thermal Resistance	36
4.4. Modelling	38
4.5. Prototype	39
5. MECHANICAL MODULE	
5.1 Research information	40
5.2 State of art	40
5.3 Choose materials	41
5.4 Choose the components	41
5.5 Define the final frame	42
5.6 Make a 3D model	42
5.7 Build the frame	43
6. MANAGEMENT	
6.1 Define Phases and tasks	44
6.2 Define deliverables and assign them to the tasks	45
6.3 Define Milestones	45
6.4 Assign the dates and resources to the tasks	45
6.5 Risks	45
6.6 Get the results	46
7. CONCLUSION	48
8. BIBLIOGRAPHY	49
9. Appendix	50

1. INTRODUCTION

- 1.1 Primes
- 1.2 Original assignment from Enit
- 1.3 Clients and requirements
- 1.4 Scope

1.1 PRIMES

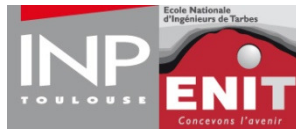
This report is about the current progress made by the Primes Project Group consisting of:

- Alex van den Biggelaar
- Javier Pérez
- Diana Sant Hernando
- Mateu Vallès Gómez

Our technical supervisor is Mr. Paul Vidal and our management supervisor is Mr. Thierry Desmaison.

This project is given by us from the ENIT in collaboration with the Primes Group. The ENIT is short for Ecole Nationale d'Ingénieurs du Tarbes, France. In Tarbes the university is located and so is our office and laboratory. The Primes Group is a platform of different companies that have similar business interests and more or less exchange their resources to improve the sector.

1.2 ORIGINAL ASSIGNMENT FROM ENIT



European Project Semester

PROJECT OUTLINE

Project dates: October – February 2011	
Title: Multilevel inverter	
Project activity areas <i>Power Electronics</i>	Subjects Static converter, multilevel inverter, prototyping
Tutor's name and coordinates Industrial tutor : To be defined ENIT Supervisor : Paul-Etienne VIDAL Paul-etienne.vidal@enit.fr	Project origin To be defined
Project technical background: The PRIMES laboratory is a common platform shared by academics and industrialists around power electronics problems such as high temperature packaging or power electronics reliability. Because of its new development, new equipments have to be defined. In the field of power electronics, a multilevel inverter has to be designed. The main objective of this EPS project is to design the multilevel inverter. Some tasks could be :	

1. Choice of the power converter architecture,
2. Simulation of the system,
3. Choice of components,
4. Prototype,

Project dates: October – February 2011

Title: Multilevel inverter

Project activity areas

Power Electronics

Subjects

Static converter, multilevel inverter,
prototyping

This subject will take place inside the PRIMES laboratory.

Study topics :

The following topics may be treated within the project :

- Power electronics
- Simulation,
- Industrial informatics, microcontroller.

1.3 CLIENTS AND REQUIREMENTS

Our clients are the ENIT and the Primes Group, they require a prototype of a 1kV*A multilevel inverter. And with the prototype they require a datasheet and manual. And the ENIT requires two written reports and two presentations.

1.4 SCOPE

After knowing the requirements from our clients we started to work out what the requirements meant for us. We figured out that we have to produce a 1kV*A multilevel inverter, that has 100V input, that is provided by the ENIT, and a load that can conduct 10A, also provided by the ENIT.

Our responsibilities lie with producing an operational rectifying electronic structure that can dissipate its own heat, have an external control module to create the sinusoidal wave output, and all of this should fit in a box so that is a one piece prototype.

Our starting budget is €300,- for the parts and assembly.

This is the final report, after sixteen weeks of working on the project in this document you will find all the information about every phase. Along with this document go the user manual, the data sheet and the appendix where you could find all specific information about primes project group.

2. POWER MODULE

- 2.1 Research information
- 2.2 State of art
- 2.3 Choose the structure
- 2.4 Choose the components
- 2.5 Simulation circuit in PSIM software
- 2.6 Design the circuit to print circuit board
- 2.7 Print the Circuit board
- 2.8 ensemble the power components
- 2.9 Power Tests

2.1 RESEARCH INFORMATION

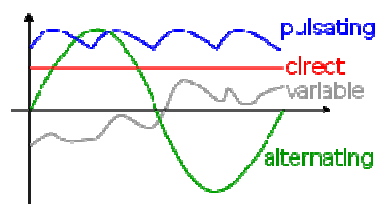
What is an inverter? And a multilevel inverter? (in power electronics)

We are starting explain what is a DC current. This is the unidirectional flow of electric charge. It is produced by such sources as batteries, thermocouples, solar cells, and commutator-type electric machines of the dynamo type.

Direct current may flow in a conductor such as a wire, but can also flow through semiconductors, insulators, or even through a vacuum as in electron or ion beams. The electric charge flows in a constant direction.

And the alternative current the movement of electric charge periodically reverses direction. AC is the form in which electric power is delivered to businesses and residences.

The usual waveform of an AC power circuit is a sine wave. In certain applications, different waveforms are used, such as triangular or square waves.



An inverter is an electrical device that converts direct current (DC) to alternating current (AC); the converted AC can be at any required voltage and frequency with the use of appropriate transformers, switching, and control circuits.

Summary of types:

- Modified sine wave:
 - Similar to a square wave output but the output goes switching positive or negative.
 - Simple and low cost
 - Not compatible with sensitive or specialized equipment

- Efficiency 80%
- Pure sine wave:
 - Produces a nearly perfect sine wave output (<3% total harmonic distortion)
 - Complex and costs
 - Compatible with all AC electronic devices
- Grid tie inverter:
 - It is a sine wave inverter designed to inject electricity into the electric power distribution system.

Inverter Applications:

- DC power source utilization: DC->AC, micro-inverters (solar panels into AC for the electric grid)
- Uninterruptible power supplies (UPS): Uses batteries (rectifier supplies DC power to recharge it) and inverter to supply AC power
- Induction heating: Inverters convert low frequency main AC power to higher frequency for use in induction heating. Method AC->DC->AC.
- HVDC power transmission
- Variable-frequency drives
- Electric vehicle drives
- Air conditioning
- The general case

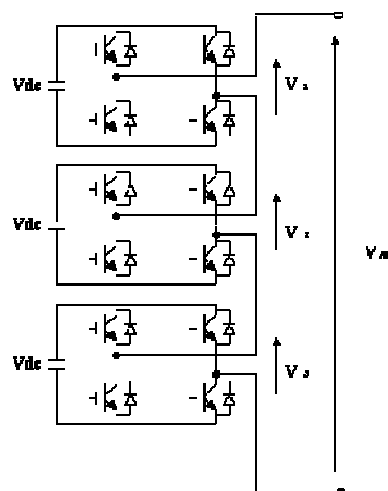
What does multilevel mean?

Multilevel means that the inverter has more than one group of switches.

Provide another approach to harmonic cancellation.

Provide an output waveform that exhibits multiple steps at several voltage levels

Example of a multilevel inverter:



2.2 STATE OF THE ART

For the state of the art we looked for the main multilevel topologies in the industry and these are classified into three categories: diode clamped inverters, flying capacitor inverters, and cascaded inverters. After these, other structures can be derived.

We describe them writing the point out good and bad of such structure and de possible applications.

Note that the different structures can be implemented for rectifying operation as well.

a) Cascaded H-bridges converter with separate dc sources.

Each inverter level can generate three different voltage outputs, $+V_{dc}$, 0 , and $-V_{dc}$ by different combinations of the four switches, S_1 , S_2 , S_3 , and S_4 .

The number of output phase voltage levels m in a cascade inverter is $m = 2s+1$, where s is the number of separate dc sources.

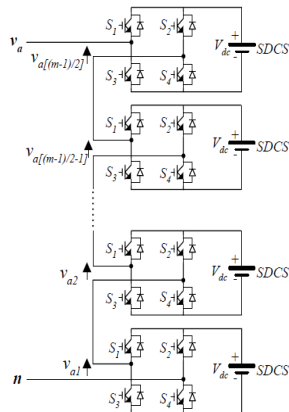


Figure 31.1. Single-phase structure of a multilevel cascaded H-bridges inverter.

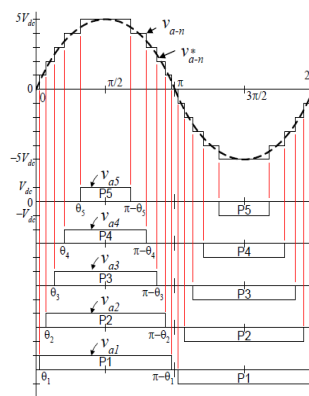


Figure 31.2. Output phase voltage waveform of an 11-level cascade inverter with 5 separate dc sources.

Advantages:

- The number of possible output voltage levels is more than twice the number of dc sources ($m = 2s+1$).
- The series of H-bridges makes for modularized layout and packaging. This will enable the manufacturing process to be done more quickly and cheaply.

Disadvantages:

- Separate dc sources are required for each of the H-bridges. This will limit its application to products that already have multiple SDCSs readily available.

Applications:

- Static var generation
- Interface with renewable energy sources
- Battery-based applications

b) Diode clamped inverter topology.

A single-phase six-level diode-clamped inverter is shown in following figure.

Each of the two phases of the inverter shares a common dc bus, which has been subdivided by five capacitors into six levels.

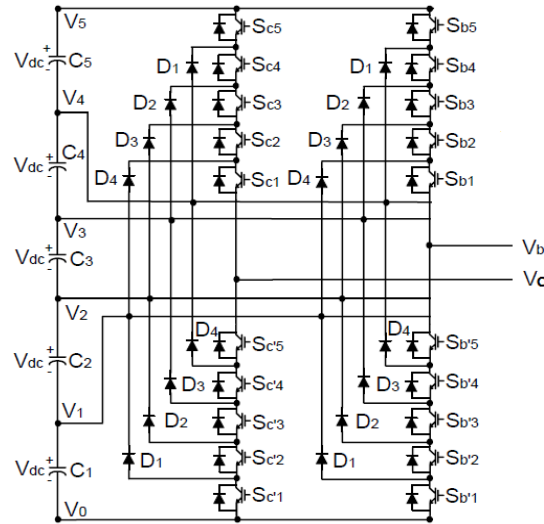


Figure 31.3 Diode clamped inverter topology

The voltage across each capacitor is V_{dc} . The voltage stress across each switching device is limited to V_{dc} through the clamping diodes.

Table 31.1 lists the output voltage levels possible for one phase of the inverter with the negative dc rail voltage V_0 as a reference. (State condition 1 means the switch is on, and 0 means the switch is off.)

Each phase has five complementary switch pairs such that turning on one of the switches of the pair require that the other complementary switch be turned off. For phase leg "a" are $(S_{a1}, S_{a'1})$, $(S_{a2}, S_{a'2})$, $(S_{a3}, S_{a'3})$, $(S_{a4}, S_{a'4})$, and $(S_{a5}, S_{a'5})$.

Table 31.1. Diode-clamped six-level inverter voltage levels and corresponding switch states.

Voltage V_{a0}	Switch State									
	S_{a5}	S_{a4}	S_{a3}	S_{a2}	S_{a1}	$S_{a'5}$	$S_{a'4}$	$S_{a'3}$	$S_{a'2}$	$S_{a'1}$
$V_5 = 5V_{dc}$	1	1	1	1	1	0	0	0	0	0
$V_4 = 4V_{dc}$	0	1	1	1	1	1	0	0	0	0
$V_3 = 3V_{dc}$	0	0	1	1	1	1	1	0	0	0
$V_2 = 2V_{dc}$	0	0	0	1	1	1	1	1	0	0
$V_1 = V_{dc}$	0	0	0	0	1	1	1	1	1	0
$V_0 = 0$	0	0	0	0	0	1	1	1	1	1

The Figure 31.6 shows one of the two line-line voltage waveforms for a six-level inverter. The line voltage V_{ab} consists of a phase-leg a voltage and a phase-leg b voltage.

The resulting line voltage is an 11-level staircase waveform.

Each active switching device is required to block only a voltage level of V_{dc} , the clamping diodes require different ratings for reverse voltage blocking. As an example, when all the lower switches $S_{a'1}$ through $S_{a'5}$ are turned on, D_4 must block four voltage levels, or $4V_{dc}$. Similarly, D_3 must block $3V_{dc}$, D_2 must block $2V_{dc}$, and D_1 must block V_{dc} . If each blocking diode has the same voltage rating as the active switches, D_n will require n diodes in series.

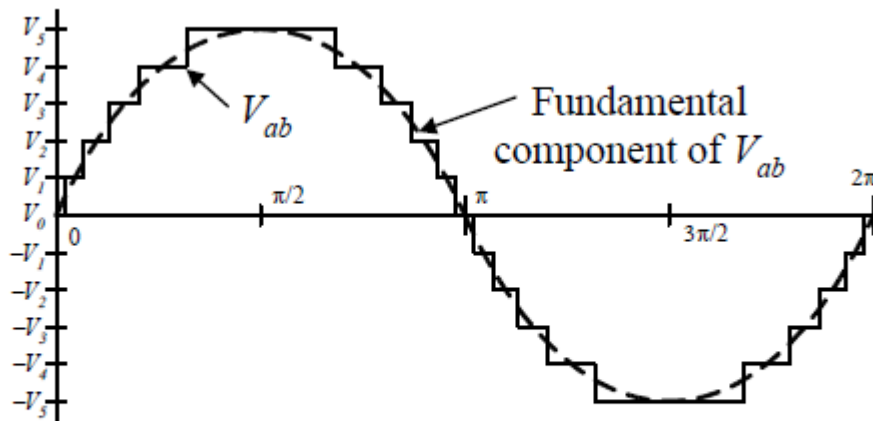


Figure 31.6. Line voltage waveform for a six-level diode-clamped inverter.

Advantages:

- All of the phases share a common dc bus, which minimizes the capacitance requirements of the converter. For this reason, a back-to-back topology is not only possible but also practical for uses such as a high-voltage back-to-back inter-connection or an adjustable speed drive.
- The capacitors can be pre-charged as a group.
- Efficiency is high for fundamental frequency switching.

Disadvantages:

- Real power flow is difficult for a single inverter because the intermediate dc levels will tend to overcharge or discharge without precise monitoring and control.
- The number of clamping diodes required is quadratically related to the number of levels, which can be cumbersome for units with a high number of levels.

Applications:

- As static var compensation,
- High-voltage system interconnections
- As an interface between a high-voltage dc transmission line and an ac transmission line
- As a variable speed drive for high-power medium-voltage (2.4 kV to 13.8 kV) motors

c) Flying capacitors

In a few words to flying capacitors inverter is to say that is the main components after the triacs are the capacitors as you can see in the figure 31.7.

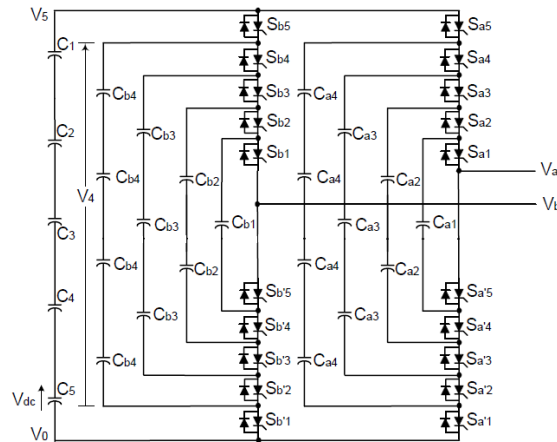


Figure 31.7. Single-phase six-level structure of a flying capacitor inverter.

Advantages:

- It has redundancies for inner voltage levels. (two or more valid switch combinations can synthesize an output voltage, in the table shows a list of all the combinations)
- The flying-capacitor inverter does not require all of the switches that are on (conducting) be in a consecutive series.
- The flying-capacitor inverter has phase redundancies, whereas the diode-clamped inverter has only line-line redundancies. These redundancies allow a choice of charging/discharging specific capacitors and can be incorporated in the control system for balancing the voltages across the various levels.
- Phase redundancies are available for balancing the voltage levels of the capacitors.
- The large number of capacitors enables the inverter to ride through short duration outages and deep voltage sags.

Disadvantages:

- Control is complicated to track the voltage levels for all of the capacitors. Also, precharging all of the capacitors to the same voltage level and startup are complex.
- Switching utilization and efficiency are poor for real power transmission.
- The large numbers of capacitors are both more expensive and bulky than clamping diodes in multilevel diode-clamped converters. Packaging is also more difficult in inverters with a high number of levels.

Applications:

- static var generation

2.3 CHOOSE THE STRUCTURE

The main multilevel topologies have been classified into three categories: diode clamped inverters, flying capacitor inverters, and cascaded inverters as we have mention in the point 2.2. In a three-phase inverter system, the number of main switches of each topology is equal. Comparing with the number of other components, for example, clamping diodes and dc-link capacitors having the same capacity per unit, diode clamped inverters have the least number of capacitors among the three types but require additional clamping diodes. Flying capacitor inverters need the most number of capacitors. And cascaded inverters are considered as having the simplest structure but with more cost.

For that characteristic we decided to use a diode clamped structure.

Analyzing the topology chosen:

The general structure of the multilevel inverter is to synthesize a sinusoidal voltage from several levels of voltages, typically obtained from capacitor voltage sources. The so-called “multilevel” starts from three levels. A three-level inverter, also known as a “neutral-clamped” inverter, consists of two capacitor voltages in series and uses the centre tap as the neutral. Each phase leg of the three-level inverter has two pairs of switching devices in series. The centre of each device pair is clamped to the neutral through clamping diodes, see figure 5.1.

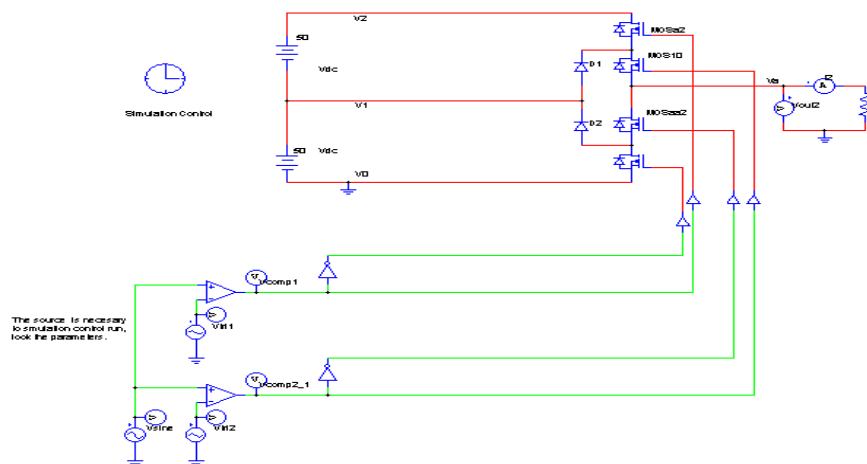


Figure 5.1. PSIM Simulation of three level diode clamped

The diode clamped inverter, particularly the three-level of this project, has drawn much interest in motor drive applications because it needs only one common voltage source. Also, simple and efficient PWM algorithms have been developed for it, even if it has inherent unbalanced dc-link capacitor voltage problem.

Considering the trade-offs between the number of levels and the voltage rating of the devices will generally lead the designer to choose an appropriate value for each.

Considering the three-level converter in Figure 5.3, connected to voltage level V_1 are the anode of D1 and the cathode of D2. D1 must be able to block V_{dc} , and D2 must block V_{dc} ; the sum of their voltage blocking capabilities is $2V_{dc}$.

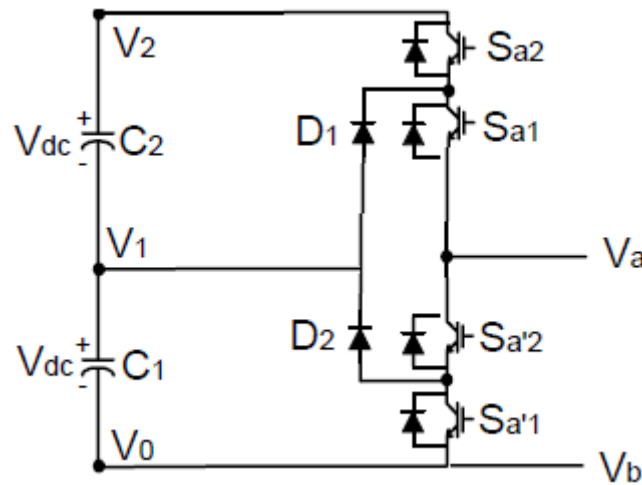


Figure 5.3. Three-level converter.

Therefore, the total voltage blocking capability per phase of an m -level converter is $(m-2)(m-1)V_{dc}=2V_{dc}$ as you can see in table 5.1.

Voltage V_{a0}	Switch State			
	S_{a2}	S_{a1}	$S_{a'2}$	$S_{a'1}$
$V_2 = 2V_{dc}$	1	1	0	0
$V_1 = V_{dc}$	0	1	1	0
$V_0 = 0$	0	0	1	1

Table 5.1. Diode-clamped three level inverter voltage levels and corresponding switch states

- The nominal RMS voltage rating, V_{nom} , of the electrical system to which the diode clamped power conditioner will be connected. The dc link voltage must be at least as high as the amplitude of the nominal line-neutral voltage at the point of connection, $\sqrt{2}V_{nom}$.
- Increasing the number of levels does not affect the total voltage blocking capability of the active devices in each phase leg because lower device ratings can be used.

Some of the benefits of using more than the minimum required number of levels in a diode clamped inverter are as follows:

1. Voltage stress across each device is lower. Both active devices and dc link capacitors could be used that have lower voltage ratings (which sometimes are much cheaper and have greater availability).
2. The inverter will have a lower EMI because the dV/dt during each switching will be lower.

3. The output of the waveform will have more steps, or degrees of freedom, which enables the output waveform to more closely track a reference waveform.
4. Lower individual device switching frequency will achieve the same results as an inverter with a fewer number of levels and higher device switching frequency. Or the switching frequency can be kept the same as that in an inverter with a fewer number of levels to achieve a better waveform.

The drawbacks of using more than the required minimum number of levels are as follows:

1. Six active device control signals (one for each phase of the parallel inverter and the series inverter) are needed for each hardware level of the inverter – i.e., $6(m-1)$ control signals. Additional levels require more computational resources and add complexity to the control.
2. If the blocking diodes used in the inverter have the same rating as the active devices, their number increases dramatically because $6(m-2)(m-1)$ diodes would be required for the back-to-back structure.

2.4 CHOOSE THE COMPONENTS

Specifications:

Our inverter should switch 100V max and 10A maximal output current. The power inverter is 1kVA.

Components of the power electronics:

- Mosfets
- Clamping Diodes
- Power electrical Wires
- Connectors
- Current sensor
- Protector devices (input fuse)

How we have calculated high power components:

- Mosfets.

We know that the voltage is 100V and the current will be 10A, we decided to use two parallel mosfet to divide the power dissipation to comply with safety regulations and we find a mosfet n.channel logic FET with 12A and 100V.

Component	Code
Mosfet	295-703
	RFP12N10L 12A
	100V

- Number of levels and voltage rating of active devices.

$$V_{\text{nom}} = 100\text{V}$$

$$V_{\text{device rating}} \cdot (m - 1) \geq V_{\text{nom}} \cdot D_{\text{design factor}} = 100\text{V} \cdot 1.5 = 150\text{V}$$

$$\frac{150\text{V}}{V_{\text{nom}}} = 1.5 \text{ levels of voltage}$$

We have decided to use three levels for the specifications and we can see the calculus look us that is enough.

-Increasing the number of levels does not affect the total voltage blocking capability of the active devices in each phase leg because lower device ratings can be used. Some of the benefits of using more than the minimum required number of levels in a diode clamped inverter are as follows:

1. Voltage stress across each device is lower. Both active devices and dc link capacitors could be used that have lower voltage ratings (which sometimes are much cheaper and have greater availability).
2. The inverter will have a lower EMI because the dV/dt during each switching will be lower.
3. The output of the waveform will have more steps, or degrees of freedom, which enables the output waveform to more closely track a reference waveform.
4. Lower individual device switching frequency will achieve the same results as an inverter with a fewer number of levels and higher device switching frequency. Or the switching frequency can be kept the same as that in an inverter with a fewer number of levels to achieve a better waveform.

The drawbacks of using more than the required minimum number of levels are as follows:

1. Six active device control signals (one for each phase of the parallel inverter and the series inverter) are needed for each hardware level of the inverter – i.e., $6 \cdot (m-1)$ control signals. Additional levels require more computational resources and add complexity to the control.
2. If the blocking diodes used in the inverter have the same rating as the active devices, their number increases dramatically because $6 \cdot (m-2) \cdot (m-1)$ diodes would be required for the back-to-back structure.

- Number and voltage rating of clamping diodes.

$(m - 1) \cdot (m - 2)$ Clamping diodes are required for an m -level converter with the same voltage rating as the active devices.

D2 must block $2V_{dc}$

D1 must block $1V_{dc}$

$$V_{\text{clamp total}} = 1 \text{ leg} \cdot (m - 2) \cdot (m - 1) \cdot V_{dc} = 2 \cdot V_{dc} = 2 \cdot 75V = 150V$$

Per unit Voltage Rating	Blocking Voltage Required	Voltage Rating of Diode Used	Nº Diodes Per leg
1Vdc	75V	75V	1
2Vdc	150V	75V	2
			Total: 3

- Power electrical Wires.

The normative Electricity of France recommend the following table:

1.5mm	10A
2.5mm	15A
4mm	20A
6mm	25A

With $5A/mm^2$ will be enough chosen 4mm of section.

APPLICATIONS:

- Motor drive

2.5 Simulation circuit in PSIM software

Thanks to this software can try to do different tests to validate the prototype.

In the following figure 2.51 you can see a structure using a three level diode clamped.

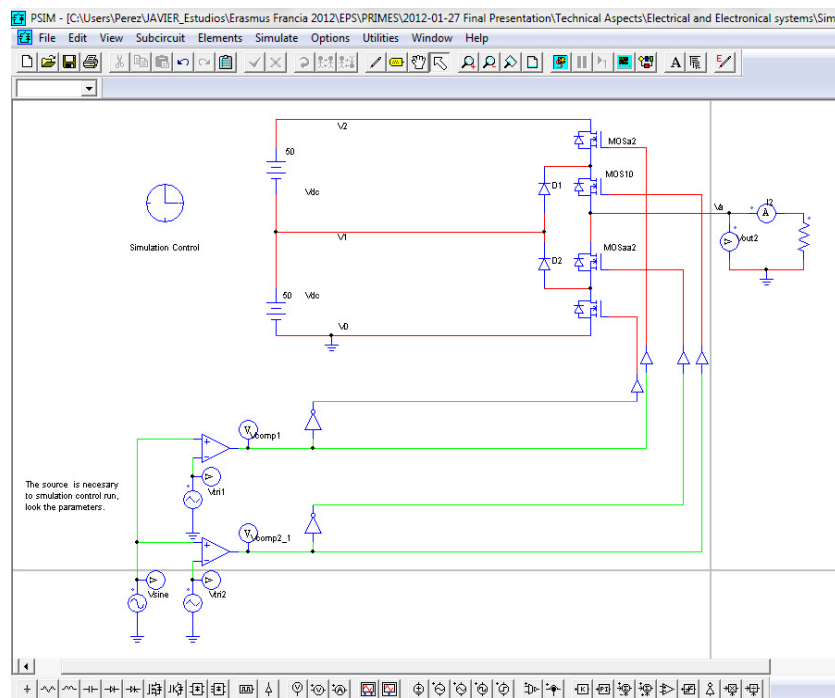


Figure 2.51 Three level diode clamped simulation in PSIM software.

The waveform obtained from a three-level inverter is a quasi-square wave output if fundamental frequency switching is used as we can see in figure 2.52.

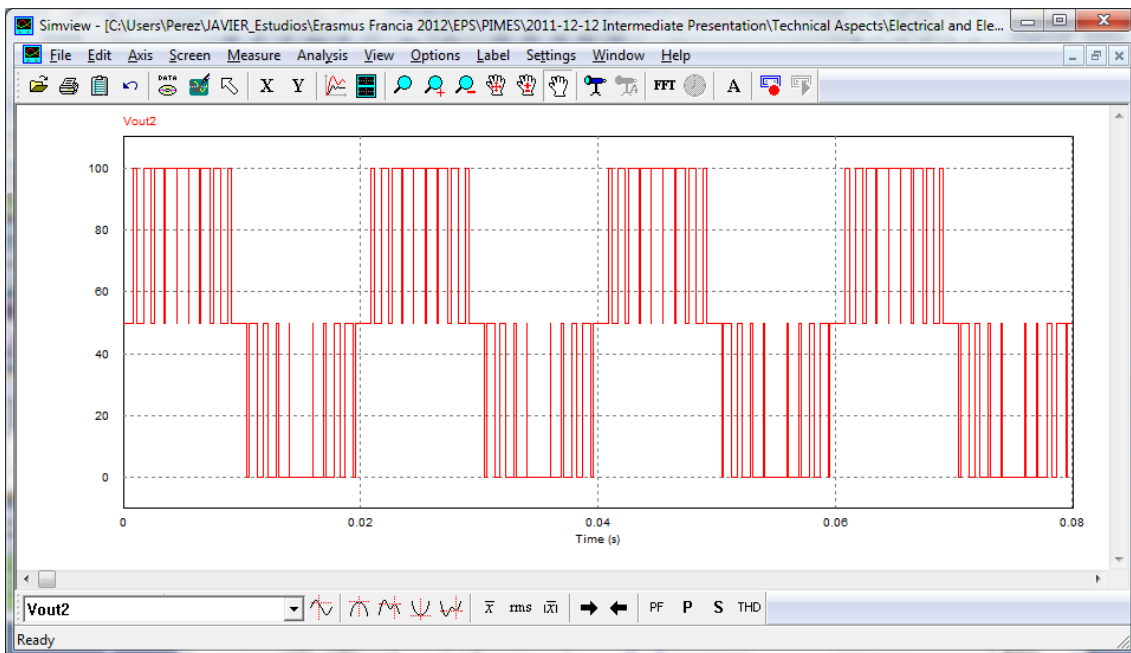


Figure 2.52 Output simulation of three level diode clamped inverter in PSIM software.

2.6 Design the circuit to Print the Circuit Board

You could start to design a power electronic circuit in several types of software. Usually some people use to Pspice and Layout to find the PCB require but there are a mount of them. A possibility that our supervisor give us was to use EAGLE software to receive help in case necessary of the technician.

Have chosen Eagle software and after learn the running we work to achieve our goal. In a first time we design the power circuit schematic. Four PCB and routing them but before print them we won't see necessary to do that and decided make only one PCB for the control components with their supplies.

We explain a little bit this process although the correct place will be in the control module but we decided put in the power module because the responsible of this module was in charge of his building and assemble.

Follow you can see a bit pictures that explain this process.

First of all the schematic view (figure2.61.a) in eagle software and the board view (figure2.61.b).

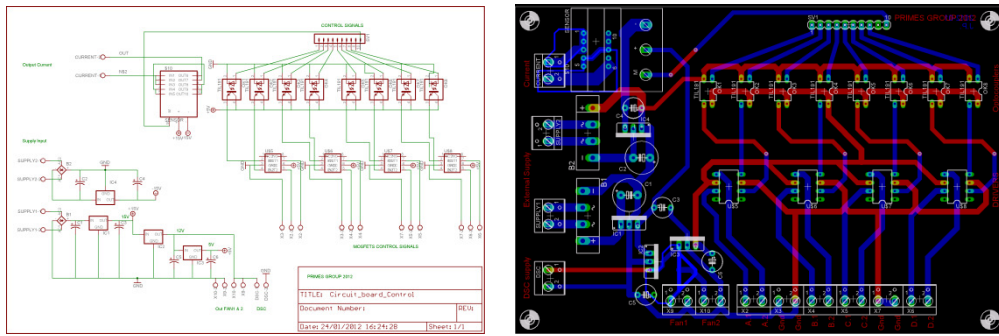


Figure 2.61 a. Schematic view and b. Board view.

Later the process to obtain the PCB, When the layout is done, the board layers are printed onto special toner transfer paper with a laser printer. This board "image" is transferred to the bare copper board with a laminating machine, or a hot clothes iron. See figure 2.62.



Figure 2.62 Laminating machine.

After laminating, the board with the paper stuck to it is soaked to remove the paper, leaving only the toner behind. See figure 2.63.



Figure 2.63 soak the PCB.

Above is a photo of the raw copper board with toner remaining, after the transfer paper has been soaked off.

Inside the etch tank and after, the toner is removed with solvent and the board is tinned using a soldering iron and a small piece of tinned solderwick. Tinning isn't absolutely necessary but it improves the appearance of the board, and prevents the copper from oxidizing before it's time to solder the parts to the board.

At this point, holes are drilled for our leaded components and mounting holes like you can see in the picture 2.64.



Figure 2.64 Drilling PCB.

2.7 Ensemble the power components

To the ensemble the power components we didn't use PCB. Instead of that we use wires like show the following figure 2.71 to make the connections required.

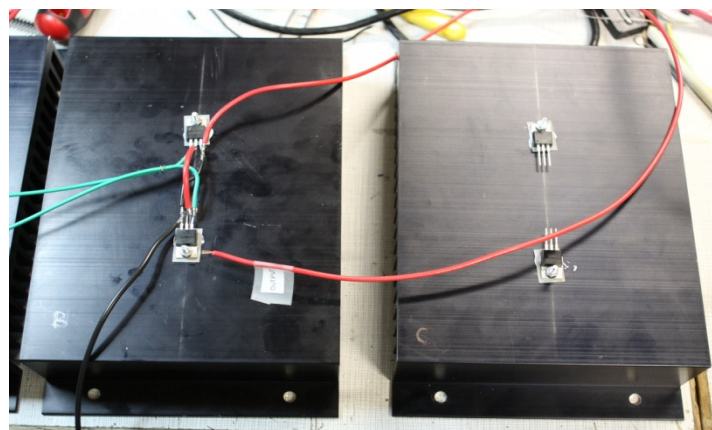


Figure 2.71 Weling the mosfets.

And like a final view in figure 2.72.

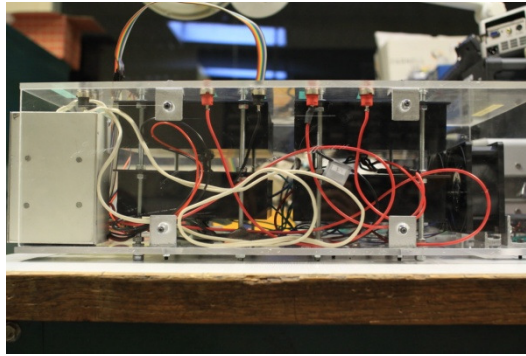


Figure 2.72 Final view of the different connections.

But to ensemble the control components, regulators and the other component necessary to have the correct supply we used the PCB design explained. Then I will explain this process with a quick figures 2.73a, b and c where we can find the PCB finished completely.

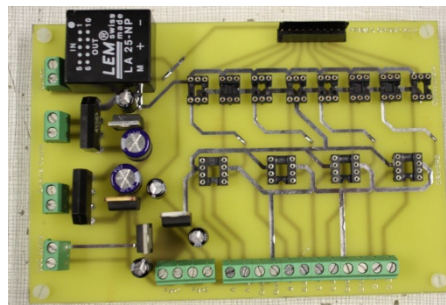
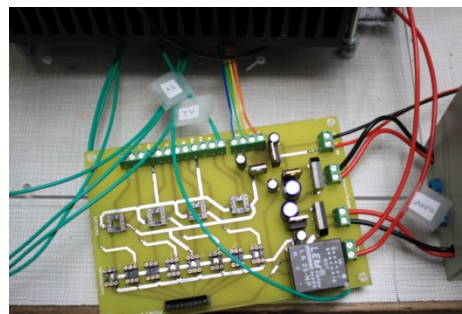


Figure 2.73 a. Weling components. b. Assembling board and c. Final view.

2.8 Power Tests

To realize the power tests would be necessary use simple software to make to run the several mosfet that conform this module but this was not possible. Instead of that we tested the joins and connections done and the Results of this have been **POSITIVE**.

For the other hand, we have done the test necessary to the PCB that follow and show with pictures the results:

Input supply after the alternative wave transformer. Our value theoretical was 18Vac and the result is 17.291Vac. See figure 2.81.

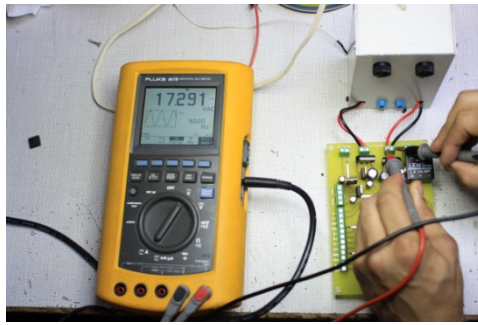
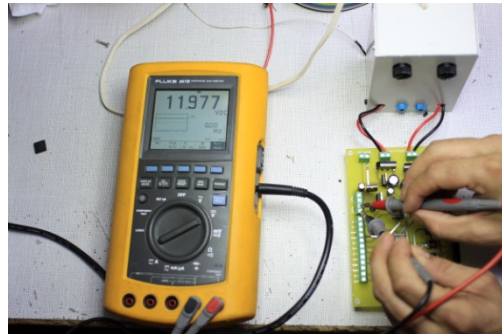
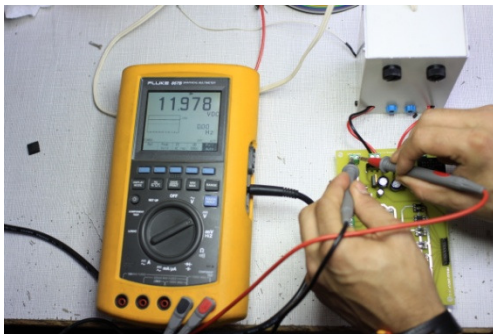
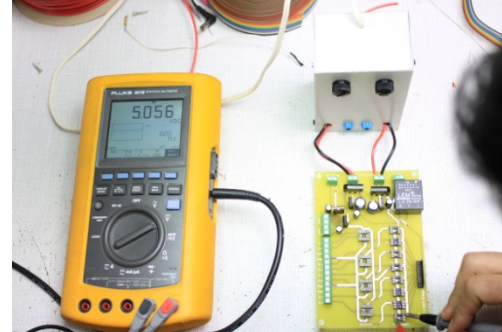
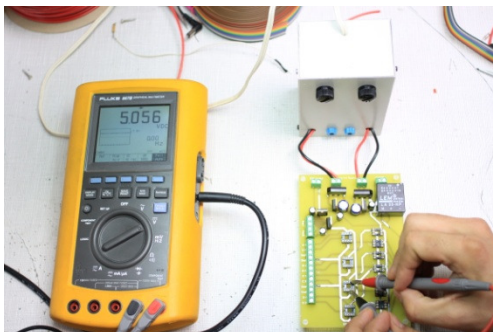


Figure 2.81 Test AC supply voltage.

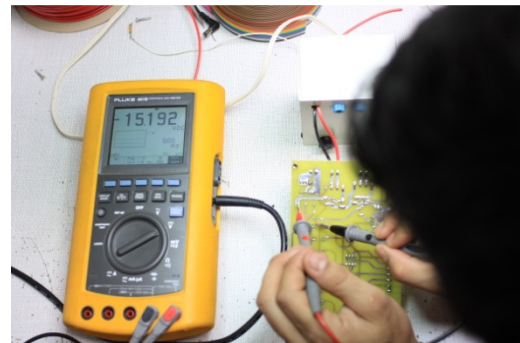
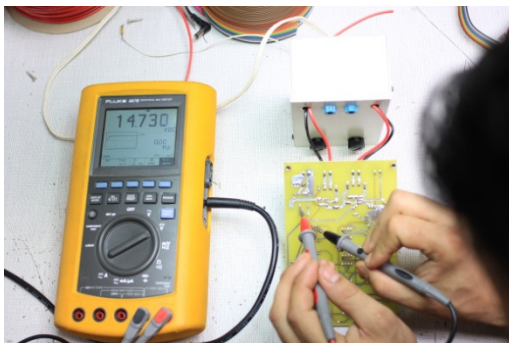
The second tests were the supply for the fans and the DSC outside of the prototype to 12Vdc like we can see in the measurements:



The third tests were for the supply for the optocouplers and drivers that need 5Vdc to run normally:



And the last tests were for the current sensor supply to +15Vdc and -15Vdc and the results were 14.73Vdc and -15.192Vdc respectively.



3. CONTROL MODULE

- 3.1 Research information
- 3.2 Accommodate values
- 3.3 Choose components
- 3.5 Acquire software
- 3.5 Design C module
- 3.6 Test modules

3.1 RESEARCH INFORMATION

To start the project the first step was to figure out what type of topology and structure we were going to use as prototype. After the decision was made for the diode clamped single phase structure I was able to start brainstorming for possibilities for controlling the MOS-FET's. This led to a few options like a microcontroller or a digital signal processor. With the collaboration of Mr. Vidal we chose for a Digital Signal Controller (DSC), which combines the strong points of the microcontroller and digital signal processor. Below is the associated research document.

Choice of Digital Signal Processor

There are several types of controlling digital signals, with microcontrollers, with DSP (Digital Signal Processing), DSC (Digital Signal Controller) and FPGA (Field-Programmable-Gate-Array).



*A **microcontroller** is basically a microprocessor, RAM and a clock combined. So you could say it's a small computer. It has also various options for peripherals that are programmable so you could design the function of the microcontroller with software. Microcontrollers are often used in embedded systems.*

***DSP** is usually used to measure, filter and/or compress continuous real-time analog signals. DSP is concerned about the processing of these signals. By sampling the signal it's converted from an analog to digital signal. After processing the signal, often times it needs to be an analog signal again. So why would we use DSP? DSP provides many advantages like: data compression, error detection and correction. The use of DSP has a wide spectrum. New technologies have enabled us to specify DSP to its use, such as a more powerful general purpose, DSC.*



DSC can be considered as an hybrid component. The DSC has both characteristics of microcontrollers and DSP's. Such as the microcontroller the DSC has fast interrupt responses and offer control-oriented peripherals (PWM). More similar to the DSP, the DSC has got multiply-accumulate units (MAC). Multiply-accumulate units can perform these kinds of calculations: $a \leftarrow a + (b \times c)$.

A accumulator stores these results in a register instead of writing them in the memory, which enables the clock frequency to be higher. The DSC are currently marketed as green technologies for their potential to reduce power consumption in electric motors and power supplies, which are the main targets for application of DSC's.

FPGA's are intergraded circuits that function as a "empty computer" for the customer. You will buy the circuit and afterwards you are going to define and configure the functions of the FPGA yourself. FPGA's are commonly used when:

The software is too slow

I/O-interfaces

Signal processing

FPGA's use logic blocks to define the function of one desired cycle. In such logic blocks, as a designer state that e.g. the logic gate 1 and 2 are the inputs for the flip-flop 3. In these logic blocks the designer can combine a very complex circuit of combinational functions.

So at this point I knew that I have a DSC to control eight MOS-FET's. The following step was to think of a secure way to led these components interact with each other. So through comparing the start and end values of these components with the Power Module we could start researching for securing the components. We focused on optical isolation of the components because we were pointed in that direction. Below the associated research document.

Optical Isolation

Optocoupler

An optocoupler is an electronic device designed to transfer electrical signals by utilizing light waves to provide coupling with electrical isolation between its input and output. The main purpose of a optocoupler is to prevent high voltages or rapidly changing voltages on one side of the circuit to damaging components or distorting transmissions on the other side. The optocoupler consists out of an emitter, a closed optical channel and a photo-sensor. The emitter is usually a LED that converts the electrical input signal into light.

The closed optical channel is simply the blocking of the surrounding light, so that it wouldn't infect the signals of the emitter and collector. This can simply be a black plastic box, as long as it doesn't enable light to shine through.

The photo-sensor is the component that detects the light from the emitter. The photo-sensor can either converts this directly into electrical energy or modulate the electric current flowing from a different power supply.

Solid State Relay (SSR)

A SSR is an electronic switching device where a small input voltage controls a larger load current or voltage. An SSR is a type of optocoupler. The SSR has no moving parts hence the name solid. The output of the switch can produce either AC or DC to the load.

Many of the current SSR's use optical coupling to isolate the input from the output.

After the optical isolation the only problem was the power level of the components because still after isolation the values were too low for the MOS-FET's. This resulted in thorough research off the electronic components internet providers catalogues. Again with the help of Mr. Vidal we made the best decision. The sites used:

Finally to finish the research I studied C++ document to be able to construct a C file so that the DSC can control the Power Module

3.2 ACCOMMODATE VALUES

The research for the isolation for the components was made possible by communicating with the Power Module about the values that we would use. Beginning at the gates of the MOS-FET's we could search for the components that would ensure smooth operation of the prototype. We originally tried it from the gates to the DSC but actually it we met in the middle. The problem was that there are countless options for amplifying the DSC signal. So we worked our way from the DSC through the optical isolation and then knew 3,3Vin and 10Vgate.

3.3 CHOOSE COMPONENTS

Choosing the components after all the research wasn't that an easy task. The DSC was easy, hence being it recommended by Mr. Vidal. The driver between the optocoupler and the MOS-FET's created a few problems. One that there is an enormous source of drivers to choose from and second that most of them had small deviations which wasn't desirable. But finally we choose a driver that suited our requirements. Here is an overview of the components ordered by the Control Module.

4 x 1.5A Dual High-Speed Power MOSFET Drivers TC4427A from Microchip

Features:

- High Peak Output Current – 1.5A
- Wide Input Supply Voltage Operating Range:
 - 4.5V to 18V
- Space-saving 8-Pin MSOP and 8-Pin 6x5 DFN

2 x High Density Mounting Type Photocoupler LTV-817 Series from Liteon

Features

- Current transfer ratio
- (CTR : MIN. 50% at $I_F=5\text{mA}$, $V_{CE}=5\text{V}$)
- High input-output isolation voltage:

1 x MC56F8006DEMO: Demonstration Board for MC56F8006 Digital Signal Controller from Freescale. Features:

- Complete pin-out available including a 40 pin header compatible with all 56F80xx boards
- Supply voltage options from USB connector, direct power supply in J1 and using standard power jack
- MC56F8006 Demo board with USB connectivity
- JTAG control and debug of MC56F8006
- BDM control and debug of MC9S08JM60
- Serial Communications port ready for RS-232
- 6 LEDs, connected to PWM channels

3.4 ACQUIRE SOFTWARE

For completing the Control Module we needed software to help us make the simulations, calculations and programming. So even before the order placement we acquired Psim software to simulate the values at the gates of the MOS-FET's. And with the use of EAGLE software we were able to design and printed-circuit-boards. And last but not least, the Code Warrior IDE for programming the DSC.

3.5 Designing and testing the C software

Designing the C file started with getting familiar with the Codewarrior IDE software. The program has included a few tutorials on how to start a project. You start with a project and not a C file, because a project includes the initialization of the CPU, the main files and the targets.

The CPU, short for central processing unit, is the component that performs all the instructions of the DSC. These instructions include basic arithmetical and logical operations. You can say that is your computer who sends signals to your monitor or in this case I/O ports.

The main file is where the programmer writes the instructions in C language. Here the programmer can tell the CPU what to do. The Main file is can contain instructions what to do during normal operation e.g. count to hundred and back. If during that basic arithmetical operation a signal is received that gives new information there are several things that can happen. The new information doesn't affect the calculation or it requires a to be handled with directly. If the newly found information, in the form a signal, doesn't affect the normal routine, the routine continues. If otherwise the signal interferes, a condition can be fulfilled changing the routine. Or it can trigger an interruption causing the freeze the main routine and forcing a different routine to finish. This is called a interrupt.

When creating a new project it proved that putting the settings in the desired way proved to be a problem, being that the computer wouldn't detect the right components on the DSC and thus not be able to download the project correctly to the DSC. So instead of creating a new project form the beginning, we used the DEMO_LABS files that were included on the CD-ROM of the DSC. In the DEMO_LABS there was a project that simply emitted the LED's one after one. The project used a counter and six if statements that would compare the value of the counter to a predetermined value. If those would match the corresponding LED would emit, but only then.

```
if (count&1)
    LED1_Toggle();
if (count&2)
    LED2_Toggle();
if (count&4)
    LED3_Toggle();
if (count&8)
    LED4_Toggle();
if (count&16)
    LED5_Toggle();
if (count&32)
    LED6_Toggle();
count++ ;
```

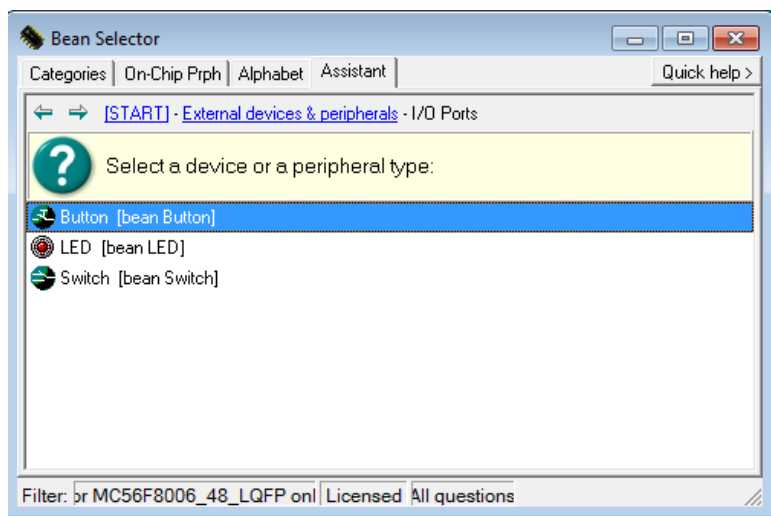
The next step was to alter the order and speed of the Toggle() function to test how to program would respond to changes. This was no problem at all and that enabled me to start doing other tests.

The second test was to create a counter that adds and subtracts. This triangle shape, would be compared to a static reference value. If the counter was below the reference value the first three LED's would be lit trough the interruption. And if the counter was above the reference value the last three LED's would be lit. The condition of the interruption was like this:

```
    If(count < RefVal)
    {
        LED1_On();
        LED6_Off();
    }
    Else If(count > RefVal)
    {
        LED1_Off();
        LED6_On();
    }
} end if;
```

Since that worked the next step was to make the static reference value variable. The purpose of this test was to create a continuous changing duty cycle, as the duty cycle in the last test was precisely 50%. The reason that we want a every changing duty cycle is to link the duty cycle's value with different output signals. For example if the duty cycle would be 35% we would like to see that two LED's would it, being 35% of 6 LED's would be two. That resulted in having a variable called counter and RefVal that both would add and subtract until two predefined values. Constantly comparing the two values and comparing that result to the number of LED's gave a variable duty cycle. All of this code was written in the interrupt.

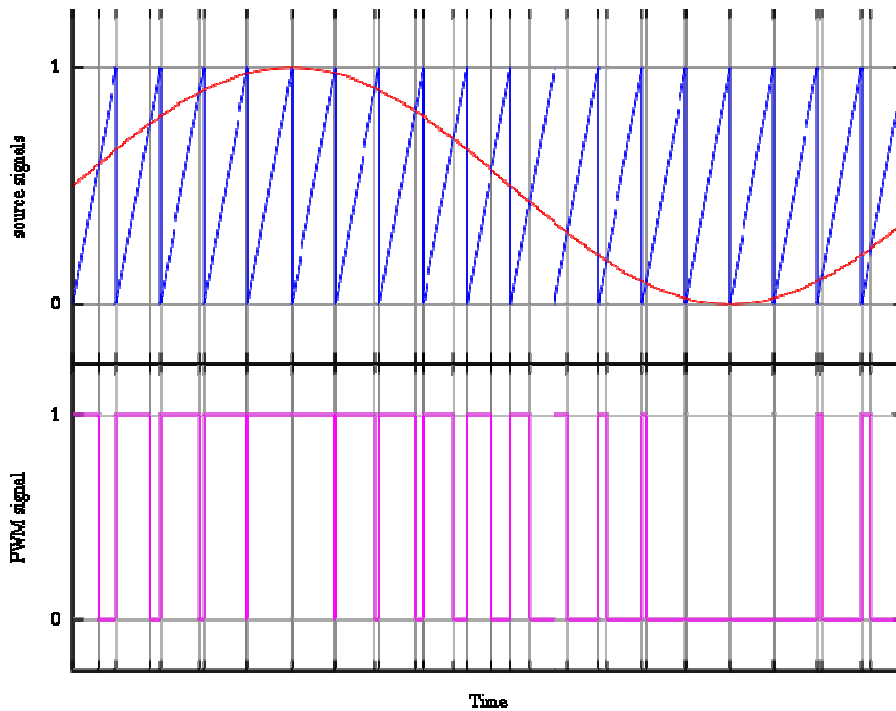
The next step was using the buttons on the board. This meant adding the button to the project. This is done in the Processor Expert view of CodeWarrior IDE. In the window shown, you can see that you can select the button with the Bean Selector. The Bean Selector is the window



where you can select all the 'Beans' of the DSC. The beans are the I/O ports, the buttons, the Timers etc. Adding a bean enables the programmer to create new

functions and difficult algorithms within a few minutes. When a new bean is added to the Processor Expert you can select which GPIO, which functions and other kind of settings. When you have chosen all of your desired settings the programmer must add the bean to the project. This is done by the 'make file' option. This option automatically adds the bean to the project by generated it's c.file and it's h.file plus of all it selected functions. With the newly added button-bean the programmer can write a routine that includes the new bean, e.g. :`void Button_Clr();` which clears the signal coming from the button. After adding two buttons to the project, I made one button1 function to alter the reference value. This means making the previous variable reference value static once again, so that the user can manually alter the duty cycle. The second button was designed to be an emergency button. If the button would be pressed the whole project would come to a stop. That means disabling the interrupts and stopping the main() routine. This was done by adding the functions in the Bean Inspector window, for all of the included beans (LED's, Buttons and Interrupts). The both buttons functions were interrupts, if one of the buttons would be pressed the button has got the highest priority and will be executed immediately. So to revise, the project now has a starting duty cycle of 50%, comparing the counter to the reference value would lit the six specific LED's and one button alters the duty cycle while the other button freezes the project. Then there is a third button which the DSC has as its own safety measure. This button restarts the whole project. So if you would have a program that count up to sixty and each time it passes ten, twenty, thirty, forty and fifty it lights up a led, let's say the counter is at 33 that means three LED's are on. If the third button would be pressed the counter will start at zero again and the LED's would turn of.

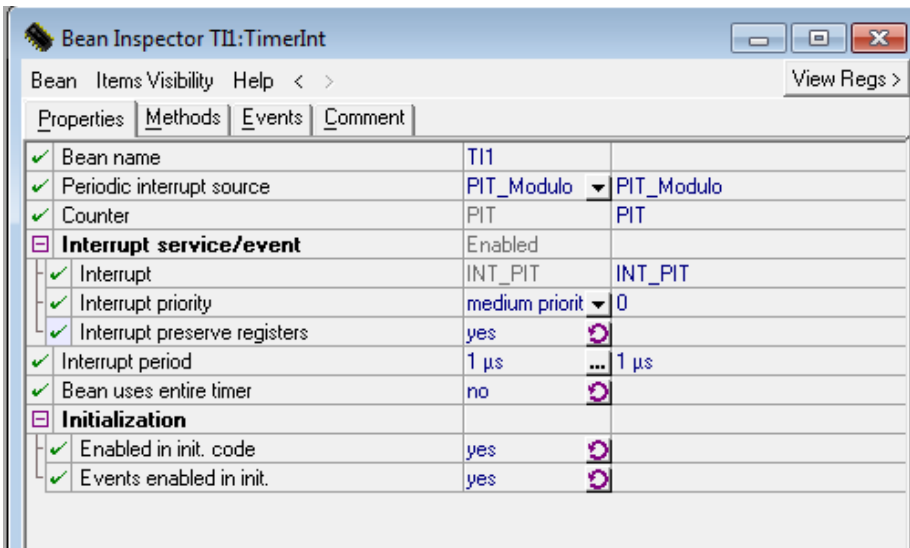
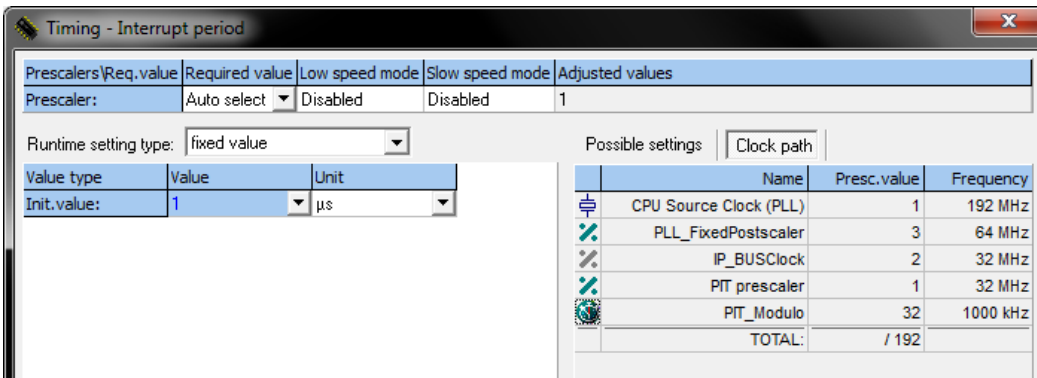
So now back to the signal we want on the output of the DSC. We need two internal signals, one triangular signal, mentioned before, and sine wave that acts as a reference value. The triangular signal is a counter that goes up and down with the help of a polarity check. If the counter reaches its maximum value an if statement would detect that and change the polarity of the counter by inverting the polarity check value. The same happens when the counter reaches its minimal value. Then the variable reference value will also go up and down in the spectrum of the counter. The middle point is the zero point of the reference value. If the reference value is above the middle point of the counter, it will send only signals to the positive side of the MOS-FET's. So if the reference value is at 78 out of scale of 100 the duty cycle signal to the positive MOS-FET's would be 56% and 0% to the negative side. In this picture it will become more clear.



As you can see the blue triangular signal is constantly compared with the red sine wave. And below the corresponding duty cycle. Now is this picture actually not perfect because we want to use a counter that isn't set to zero every time it reaches its maximum value. Why? Because we want to use the middle point as reference for positive or negative output.

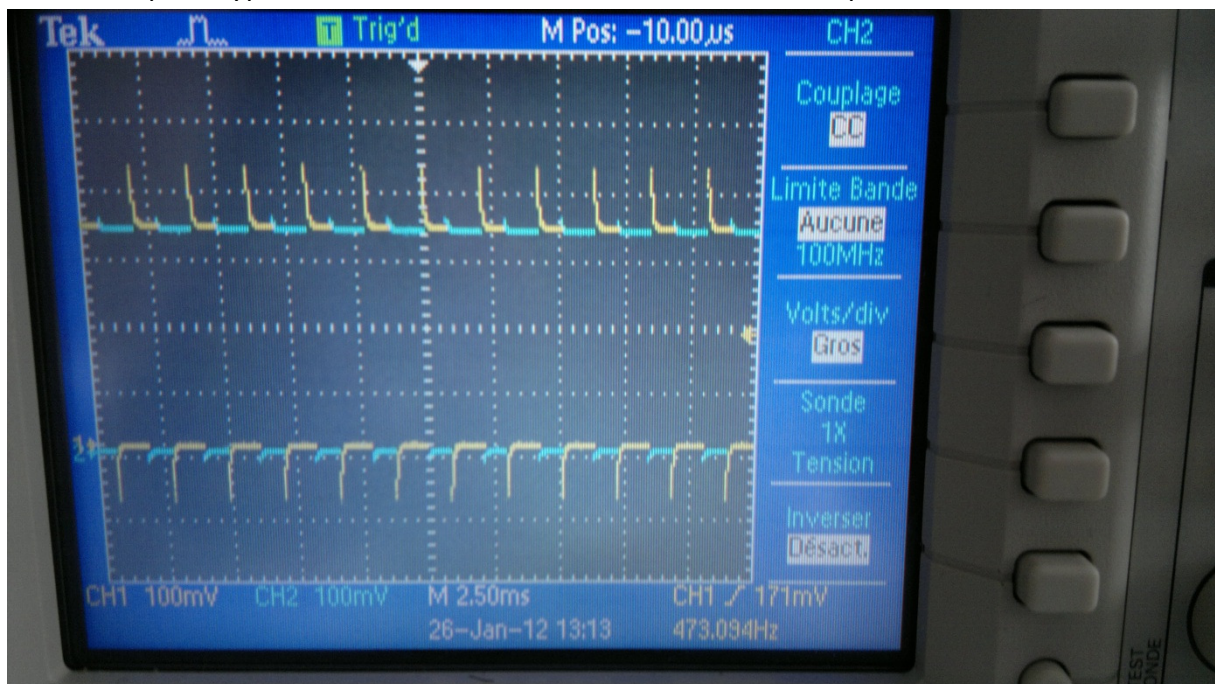
Then the testing using several small projects and compiling them to the Digital Signal Controller enabled me to test the performance of the software. I wrote my software so that all of the signals would be sent to the LED's or the GPIOE port of the DSC. The GPIOE port of the DSC is an eight-pin connector. That perfectly matched with the need for eight signals for the MOS-FET's. I used the Tektronix TDS 2014B Four Channel Digital Storage Oscilloscope that was provided by the laboratory of ENIT Tarbes. Using the number three pin on the forty-pin GPIO strip as a ground, I connected two probes to the GPIOE port. Testing the DEMO_LED_LAB first I could see that the frequency of the was nowhere near the desired 1kHz. After testing several project and files, I encountered the problem that I was not able to send the signals at the 1kHz value. This was a difficult problem. After numerous hours of trying to find out where the clock signal came was coming from and looking for the right way to write the code the solution wasn't found. So together with Mr. Vidal we together looked at this problem, but unfortunately the two of us weren't able to figure it out. The TimerInterrupt we used could be set to 1000kHz. Only whenever there was more than two lines of code within the interrupt handler the frequency would drop non-linear. Example:

```
Count + bitNegVal() >> 784kHz
Count + bitNegVal() + LED1_Toggle() >> 633kHz
Count + bitNegVal() + LED1_Toggle() + LED2_Toggle() >> 531kHz
Count + bit1NegVal() + bit2NegVal()>> 632 kHz
```



As you can see in the previous pictures the Interrupt period is supposed to be 1000kHz with these settings. What seemed to be the problem was that the Periodic interrupt source shouldn't be a TMR or PIT, but the RTI, Real Time Interrupt. The Real Time Interrupt would be able to produce signals at such frequency's, only every time it empty's it's memory. That means you can run it once. This very delicate methods were the red line in the control module.

So what finally became the software wasn't ready to be implemented as operational within the prototype. The last test of the software led to this output:



This is the measurement of the GPIOE port. The frequency wasn't able to go any higher than 423 Hz without losing if statements or signals. The blue and yellow lines represent two signals that only switch according to this if statement:

```
If(counter <= 125)
{
    Bits1_PutBit(0, 1);
    Bits2_PutBit(1,0);
}
```

All of the other tests and software will be included in the appendix.

4. Cooling Module

Now let's leave aside the electronic section of the project and let's move onto the mechanical part. As stated earlier this section will contain the cooling system of the multilevel inverter in detailed.

First of all let me start by stating the tasks related to the cooling system.

- 4.1 Why do we need a cooling system?
- 4.2 Finding the right components
- 4.3 Calculations of the Thermal Resistance
- 4.4 Modelling
- 4.5 Prototype

4.1 WHY DO WE NEED A COOLING SYSTEM?

It is important to understand the reason why we need a cooling system. This reason is that there are some electronic components that dissipate heat but those components would break at such elevated temperatures. Therefore, we looked at how we could come up with the solution and what we found was that we needed to cool the mosfets that are the ones that dissipate more heat.

The ever growing electronic industry is in a constant search for new ways to cool the components. From giant fans to liquid nitrogen, industry is continually striving for better and more quiet and reliable cooling methods.

Various techniques are currently used to cool electronic components such as Mosfets, which can reach temperatures high enough to cause permanent damage or even break if they are not kept at a safe temperature in an appropriate form.

Air cooling

Passive cooling is probably the most ancient and common method for cooling not just electronic components but anything. The idea is that a heat exchange occurs between the air at ambient temperature and the cooling element, at higher temperature.

The system is as common that is not in any way the invention of man and nature use it all the time. Then, the technique for cooling electronic components is increasing the contact surface between the air and the component in order to maximize the heat that can be removed. Just in order to maximize the contact surface, heat sinks consists of hundred of thin fins. The more fins the more heat dissipation as well as thinner the better.

Passive air cooling; the main advantages of passive heat dissipation are its simplicity (since it is basically a large piece of metal), its durability (as there are no moving parts) and its low cost. Besides the above, they don't produce any noise. The major disadvantage is its limited ability to disperse large amount of heat rapidly.

Forced air cooling; is basically taking a passive system and adding an element to accelerate the flow of air through the fins of the heat sink. This element is usually a fan but variants have been used in a kind of turbine.

In passive cooling tend to happen that the air surrounding the heat sink gets hot, and their ability to evacuate heat from the heat sink decreases. Even though natural convection moves the hot air, is much more efficient to incorporate a mechanism to force a flow of fresh air through the heat sink fins, and is exactly what forced air cooling does.

Liquid cooling

A more complex and less common is water cooling. Water has a specific heat and a better thermal conductivity than air, so it can transfer heat more efficiently and at larger distances than gas. Pumping water around the transistors can remove large amount of heat from it in a short time and then be dissipated by a radiator located somewhere near the inverter. The main advantage of liquid cooling is the ability to cool even the hottest components of the inverter.

These types of cooling methods mentioned are the ones that are commonly used and more straightforward.

4.2 FINDING THE RIGHT COMPONENTS

We decided that we were going to use forced air-cooling instead of liquid cooling because it's cheaper and therefore it fits better with our budget and it endangers less the circuit as it has no water that could be really hazardous for the electronic components.

It is quite obvious that the more surface the heat sink has the better would be its dissipation, but is also true that any intermediary in any cycle, despite how perfect it may be, is a hindrance. It's like when you add cables between a stereo and speakers, the more cables, the more signal deteriorates, and this occurs by simple entropy even if you use gold wires. So what is the benefit of using a heat sink? Why wouldn't it be better to let the electronic component to deal directly with the air? This is because in addition of the surface, involved in this phenomenon there is a property called thermal conductivity of a materials ability to channel the heat.

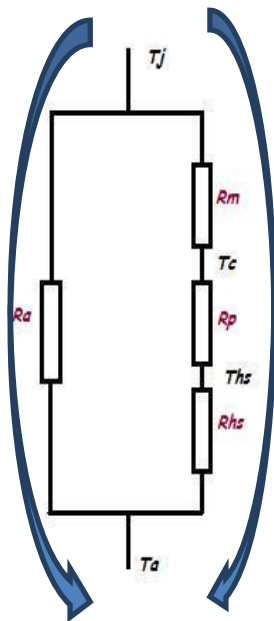
After all the research we eventually decided that the components selected were heat sinks and fans.

The modern heat sinks are usually manufactured in copper or aluminium, which are excellent conductors of heat and are relatively cheap to produce. That's the reason why we have used an aluminium heat sink so that it fits with our budget.

4.3 CALCULATIONS OF THE THERMAL RESISTANCE

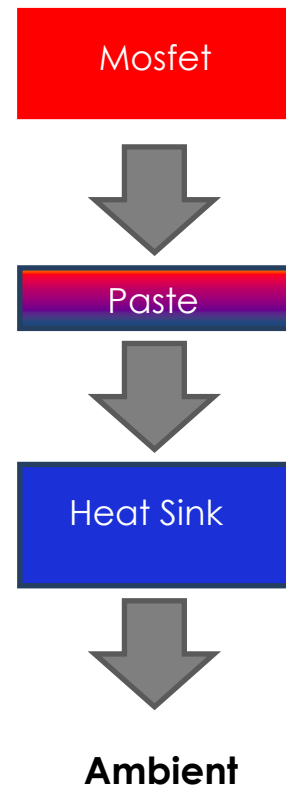
To start with the technical aspects, let's get into details about the calculations of the thermal Resistance which are needed to select the right heat sink.

In the circuit below you can see the two different paths of the heat dissipation from the mosfet to the air. Then, let me explain the components of the circuit.



Firstly I will start by the fastest path which is directly from the mosfet to the air (left hand side of the circuit). In the circuit is shown as the air resistance.

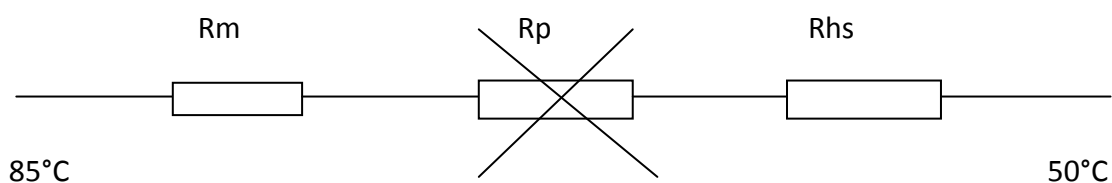
Then the other path is from the mosfet through the heat dissipation paste, then the heat sink and finally to the air (Right hand side of the circuit).



The source of the heat is the junction of the mosfet which is at 85°C and then it goes to the case of the mosfet. In the circuit appears as R_m that stands for Mosfet Resistance.

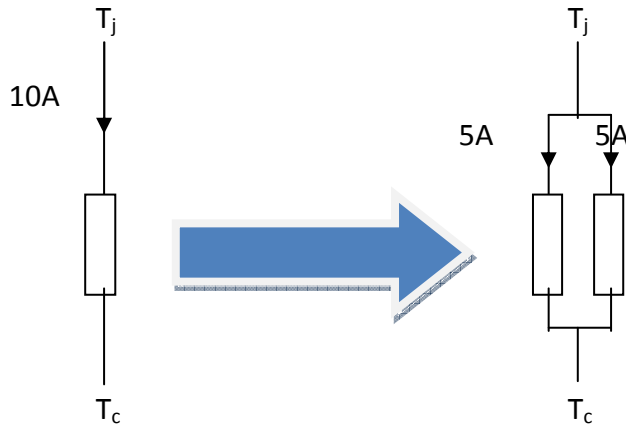
Then, the heat goes through the paste for heat dissipation to the heat sink. In the circuit appears as R_p that stands for paste resistance.

Finally, it goes through the heat sink to the air reaching a working temperature of 50°C.



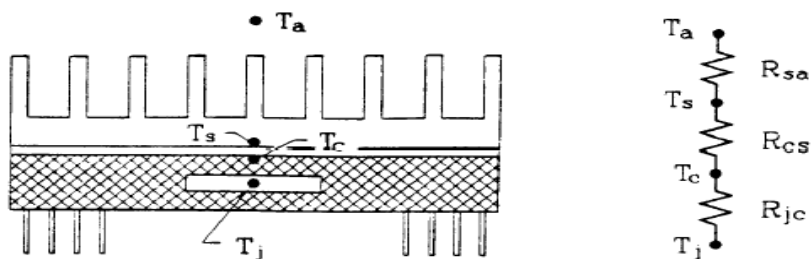
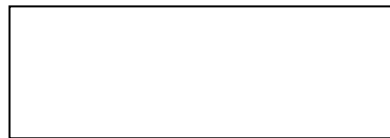
Assumptions:

- Neglect R_p as its value is so small is not worth taking into account.
- $T_a=50^\circ\text{C}$, which is going to be the working temperature inside the frame.
- We will use fans besides the heat sink.
- Current changed from 10A to 5A.



Where T_c is the Temperature of the case of the mosfet.

Equations used during the calculations



$$Pd = 2 \cdot (0.2 \cdot 1.25) \cdot (5^2) = 12.5W$$

$$R_m + R_p + R_{hs} = \frac{T_j - T_a}{Pd} \rightarrow \frac{2.085}{2} + 0 + R_{hs} = \frac{35^\circ\text{C}}{12.5W \cdot 2}$$

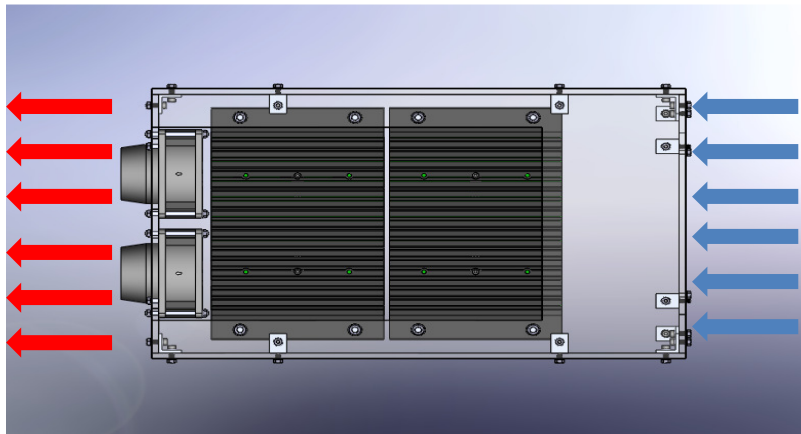
$$R_{hs} = 0.36^\circ\text{C}/W$$

After all the calculations we finally obtained that the Thermal Resistance for the Heat Sink is $0.36^{\circ}\text{C}/\text{W}$.

Before doing the calculations, we decided the components we were going to use but we still had to determine the amount of them.

Our initial idea was to use one big heat sink for the 8 mosfets, but we needed to take into account the result for the thermal resistance calculated before, so that lead us to determine that the final number of heat sinks used was going to be 4.

After that to be certain that the heat was dissipated properly and that it wasn't accumulating we installed fans. The number of fans was decided according to the dimensions of the heat sinks, to cover all the length of them. To be more specific, there were installed 2 fans.



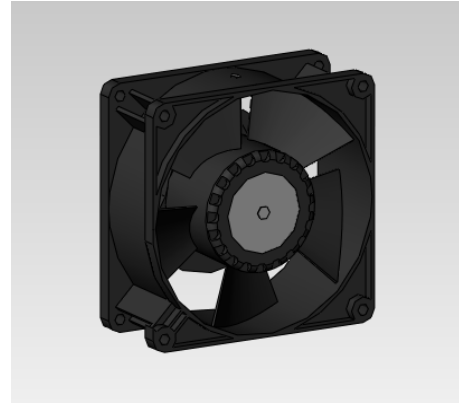
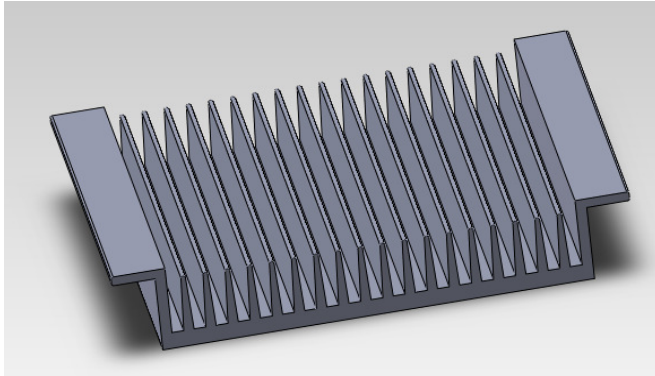
It is important to understand how the ventilation is working. Firstly the fans take air from the outside part of the frame and then the air goes through the inner part taking out the heat and releasing warm air.

The front and back are completely opened to ease the ventilation, so that the heat is dissipated properly.

4.4 MODELLING

Next, we made the 3D models of the components which allowed us to start with the 3D modelling of the whole prototype.

We used a program called Solidworks that helped us to either create the pieces or download them from a specific Solidworks library on the internet.



In this case, the heat sink was done in solidworks and the fan was downloaded.

4.5 PROTOTYPE

Furthermore, the last task was to assemble the cooling components in order to build the whole prototype.

Because of the changes done in the mechanical module, the cooling module was affected and so it had to be changed as well.

The only change that needed to be done was to join the fans to the bottom layer instead of the front layer. In the initial design we did in Solidworks the fans were attached to the front layer by screws and nuts. In the real design the fans are attached to the bottom layer by 4 L-shaped blackest and screws and nuts.

The heat sinks were placed in the same position and in the same way as in the design. We used 8 big screws to join the heat sinks with the bottom layer and with the corresponding nuts and washers.

These changes do not affect the operation of the cooling system, as the air is forced throughout the inner part of the multilevel inverter. Now there is more space for the air to enter the inner part of the frame so the flow will be smoother.

5. MECHANICAL MODULE

- 5.1 Research information
- 5.2 State of art
- 5.3 Choose materials
- 5.4 Choose the components
- 5.5 Define the final frame
- 5.6 Make a 3D model
- 5.7 Build the frame

5.1 RESEARCH INFORMATION

At the beginning of the mechanical module it was very important to have a first look at many inverters or multilevel inverters. The aim of this research was to know exactly the physical appearance of the device we want to build.

The responsible of the mechanical module with the help of another team member basically used internet for doing this task. We looked online at different technical websites. Apart from the web, our technical supervisor also facilitated us some useful documents where we could distinguish several designs of multilevel inverters.

We came across some difficulties when browsing the web. The results we obtained were sometimes really different. For example we obtained simple car inverters that use the 12V from the integrated car lighter. So this result wasn't useful as what we were looking for was high power multilevel inverters. To solve this problem we narrowed the search in order to get more precise results.

Fortunately we then obtained images of multilevel inverters for railway purpose which were indeed high power devices. About the appearance of these right multilevel inverters we can say they were big and robust. We checked that the materials used were basically metals.

5.2 STATE OF ART

This section has to do a lot with the research of information. So this task was actually done in parallel with the browsing on the web. The state of the art is a very important part when facing any project, design or investigation. The aim of the state of the art is to check the actual technology, what is used at the moment of the beginning of the project or design.

In our case, we focused on the mechanical structure of high power multilevel inverters. We realised that multilevel inverters were used for vehicle, railway, solar, nautical and informatics applications.

The conclusion after this specific research was that the devices were big, robust, made out of metal, followed their applications rules and fulfilled the security rules of each purpose.

5.3 CHOOSE MATERIALS

After the global and specific researches we were ready to decide the materials for the external frame and the fixing components. We could chose between plastic and metal for doing the frame and also for the components that will give strength to the frame.

So we decided to use acrylic transparent plastic for the frame and iron and steel for the fixing components. To make this choice we took into account the budget available and the sturdiness we wanted for the frame. We thought that thick plastic layers would fulfil this goal. We also contemplated that making the box transparent would ease the task of checking the inner cooling and electronic components.

For the fixing components we thought that any metal would give strength to the structure. Following this condition and taking into account that many screws, nuts and washers were metallic we decided to use steel.

5.4 CHOOSE THE COMPONENTS

As mentioned before we will use plastic layers for the frame. Their thickness will be 6mm to have a robust case as a result.

We decided to use iron L-shaped brackets to join the plastic layers. To fix these pieces with the layers we will use steel screws, nuts and washers.

For the inner cooling components such as the heat sinks we will use large screws that will cover the whole height of the multilevel inverter. As there are 8 big screws for attaching the heat sinks to the frame they will definitely give a lot of strength to the structure.

STEEL

- 8 big screws with no head M6x175
- 24 socket head Allen screws M4x12
- 8 socket head Allen screws M3x16
- 56 nuts M6
- 24 nuts M4
- 8 nuts M3
- 56 washers M6
- 24 washers M4
- 8 L-shaped brackets

PLASTIC

- 2 plastic layers 500x300 mm
- 2 plastic layers 300x150 mm



5.5 DEFINE THE FINAL FRAME

Taking into account all the aspects mentioned before, some done in parallel, we were able to decide the final appearance of external frame and the fixing components for the inner layout.

The frame will be a plastic parallelepiped that will measure 465x280x164mm approximately. These dimensions are slightly to be different in the real prototype due to possible changes in the inner layout.

The front plastic layer will have two squared holes for putting the fans. The back plastic layer will have the shape of a U, because we want to ease the ventilation of the box. Then the plastic underneath will not have any big holes, apart from the ones for the screws, to ensure stability. Finally, the top layer will also have the shape of a big U to let the heated air of the heat sinks go upwards.

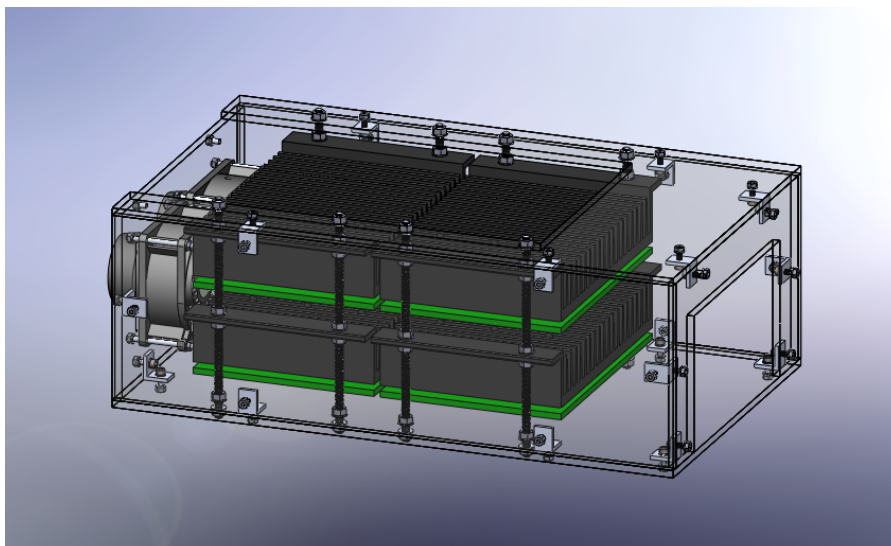
In order to fix all the layers we will use the 18 L-shaped brackets that, with the help of the screws, nuts and washers we will obtain an optimum structure.

5.6 MAKE A 3D MODEL

What helped the members in charge of the mechanical module was the engineering software Solidworks. It is a program that enables to make 3D models of any piece and also assemblies.

We made a model of each piece of the multilevel inverter with their real sizes in order to know the dimensions of the external frame. We also were able to know if the inner layout of the cooling and electronic components was feasible.

As a result we now have a file of the whole device which is no more than a big assembly of smaller assemblies. This design will be indeed a very good guide for us while making the real prototype.



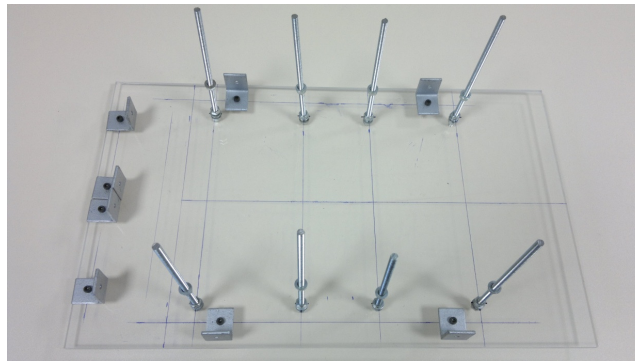
5.7 BUILD THE FRAME

We have followed the design to build the external frame, even though there are some changes in compare with the first design. The main differences are that the first design was going to need 6 plastic layers whereas in the real prototype there are only 4: The top, bottom and lateral layers.

The plastic layers were cut using a specific big saw that worked efficiently, even though it is needed to say that we had some difficulties trying to cut the layers with other smaller saws before.

After we had the four layers cut we drilled the holes for the screws that will attach the heat sinks, the fans and of course, the L-shaped brackets to join the layers.

Once the layers had being worked out we assembled the layers with the screws, the L-shaped brackets, the nuts and washers. At that point we were ready to add the other components of the other modules such as the fans, the heat sinks and the PCB.



6. MANAGEMENT

Every project needs to be managed. So, for our project we decided at first to use an excel template in order to have a management file. When we learnt how to use this template we realised that it was too simple and it did not have the characteristics we were looking for. We needed more powerful software.

During the European Project Semester all of us, the students, were taught MS Project. When we had the knowledge and the skills we decided to start a new file in this program. These are the steps we followed in order to be successful;

- 6.1 Define Phases and tasks
- 6.2 Define deliverables and assign them to the tasks
- 6.3 Define Milestones
- 6.4 Assign the dates and resources to the tasks
- 6.5 Risks
- 6.6 Get the results

Here is the description of the different steps:

6.1 DEFINE PHASES AND TASKS

The tasks are the different things that have to be done during the project. Then we have the phases, which are a group of tasks encompassed in a subject or department.

Phases	Duration in days	Person in charge
Team working	80	All
Milestones	41	All
Management	78	Diana, Mateu
Power module	64	Javi
Control module	36	Alex
Cooling module	20	Diana
Mechanical module	32	Mateu

6.2 DEFINE DELIVERABLES AND ASSIGN THEM TO THE TASKS

The deliverables are the work or documents that can be shown or delivered at some point of the project. So, in other words, a deliverable is a term used in project management to describe a tangible or intangible object produced as a result of the project that is intended to be delivered to an internal customer as ENIT or an external customer as PRIMES. A deliverable could be a report, a document, a server upgrade, or any other building block of an overall project.

6.3 DEFINE MILESTONES

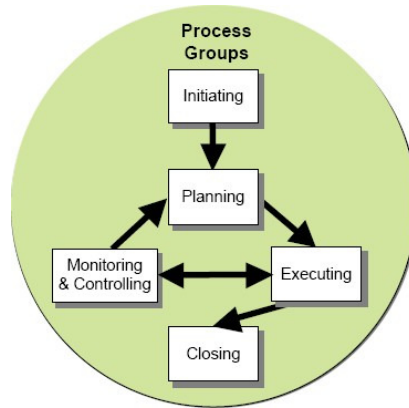
Within the framework of project management, a milestone is a special item that receives special attention. It is often falsely put at the end of a stage to mark the completion of a work package of phase. But milestones are rather to be put before the end of a phase so that corrective actions in case of problems can still be met and the deliverable can be completed in time.

6.4 ASSIGN THE DATES AND RESOURCES TO THE TASKS

The resources are the people involved in the project. In the MS project it is needed to link the resources with the tasks. Furthermore, we have to assign the dates for each task.

6.5 RISKS

After the initiation stage where the research is done, the project is planned to an appropriate level of detail. The main purpose is to plan time, cost and resources adequately to estimate the work needed and to effectively manage risk during project execution.



As with the Initiation and planning processes group, it is better to spend time in this step to make sure the Project is going in the right direction. If you do not do this correctly at the beginning you would move forward in the project but in the assembly and monitoring processes you could have problems and you would need to start the planning again.

6.6 GET THE RESULTS

Before doing the MS project file we decided to estimate the overall budget by doing some rough calculations considering the following data.

16 weeks

5 days/week

5 h/day

4 engineers

80€/h·engineer

$$Time\ worked = 14weeks \cdot 5 \frac{days}{week} \cdot 5 \frac{h}{day} \cdot 4engineers = 1400h$$

$$Time\ worked\ in\ 1day = 1day \cdot 5 \frac{h}{day} \cdot 4engineers = 20h$$

$$Budget\ 1\ day = 20h \cdot \frac{80\text{€}}{h \cdot engineer} = 1600\text{€}$$

$$\text{Total time worked} = 1400 - 2\text{days} \cdot 20\text{h} = 1360\text{h}$$

$$\text{Budget engineers} = 1360\text{h} \cdot 80\text{€/h} \cdot \text{engineer} = 108800\text{€}$$

$$\text{Prototype} = 300\text{€}$$

$$\text{Total budget} = 108800\text{€} + 300\text{€} = 109100\text{€}$$

Furthermore, it is also truly important to mention the budget when calculating the total cost of the project. We had a budget of 300€ at the beginning of the Project, but unfortunately looking at the prices when making the list of components we realized that we exceeded 300€. So, with this list we went to the client and they exceeded the budget to 400€ so that we could be able to build the prototype.

At the end of the file, the MS project provides a final document with all the result including the start and finishing date, the resources and the budget. Bear in mind that we took the work load into account the budget but also the hours worked. The price the clients agreed to pay was 80€ per hour per engineer. Adding all hours worked for the 4 engineers we get 73424€.

Now, adding the prototype money and the workload we get 73824€ as you can see in the following table;

Description	
Initial budget for components	300€
New budget for components	400€
Estimated budget for the workload of the project in MS Project	73424€
Total budget	73824€

So, as you can see there is a big different between the estimation and the final result for the total cost, luckily the real cost is lower. It is important to mention as well that the prototype cost is almost nothing in comparison with the total cost of the project.

7. CONCLUSION

To conclude this document, we have done all the necessary research to choose a direction for our project. The direction being a single phased, diode clamped 1KV*A Multilevel Inverter.

Following the research came the sketches and simulations of the casing and circuits. Most of the simulations and calculations were executed with software like Eagle, Psim and Solidworks.

After each department made their first design, the collaboration resulted in the first order of components. Validating the designs and list of components allowed each member of the team to work more specifically on their department.

The team managed to join the work from the four technical modules being successful in the assembly of the prototype.

For the engineers that want to continue this project we have the following suggestions:

- Find a good solution, software-wise, to make sure the clock speed of the DSC on the I/O output is correct. Try using the RTI-clock.
- After the new software design and tests of the software, implement it with the prototype and test the whole prototype and document this.
- If the budget allows it, you might consider making a three-phase Multilevel Inverter.
- Advice: if a second Multilevel Inverter would be build, either be careful when assembling the prototype, or use a stronger plastic material.

Bibliography:

<http://www.chw.net/2007/03/distintos-tipos-de-refrigeracion/>

<http://www.geocities.ws/djbolanos/RESUMENDISIPA.PDF>

<http://www.kabytes.com/tutoriales/como-elegir-un-cooler-para-nuestra-pc/>

SERI LEE. Advanced Thermal Engineering. Aavid Thermal Technologies, Inc. Laconia, New Hampshire 03247. How to select a heat sink.

<http://www.wikipedia.org/>

Appendix

Here are all the related documents, datasheets and information that relate to the final report.

All of the make and generated code:

```
/*
** #####
** This file was created by Alex van den Biggelaar
** for the Prototype of the Primes Project Group 2011/2012.
** Constant Duty Cycle
** #####
*/

int Check = '1';           //This variable is there to determine if we are going to count up- or downwards.
int Group1;               //This variable represents 4 MOS-FET's or LEDS.
int Group2;               //This variable represents the other 4 MOS-FET's or LEDS.
int cnt = 0;              //This variable is the counter.
int RefValue = 100;       //This is the Reference Value to determine if the output should be '1' or '0'.

int main()                 //Start routine.
{
    if (Check == '1')      //If Check is true then proceed.
    {
        Group1 = '1';      //Make Group1 true, 4 LEDS/MOSFETS are enabled.
        cnt += 1;          //Counter +1.
    }

    else                    //If Check is false then proceed.
    {
        Group2 = '1';      //Make Group2 true, the other 4 LEDS/MOSFETS are enabled.
        cnt += 1;          //Counter +1.
    }

    if (cnt == RefValue && Check == '1') //If counter has reached the Reference Value and Check is true
    proceed.
    {
        Check = '0';       //Make Check is false.
        cnt = 0;           //Reset counter.
        Group1 = '0';      //Disable Group1 LEDS.
    }

    //When this function is run, the first 4 LEDS are disabled and the second if statement is enabled.

    else if (cnt == RefValue && Check == '0')//If counter has reached the Reference Value and Check is false
    proceed.
    {
        Check = '1';       //Make Check is true;
        cnt = 0;           //Reset counter.
        Group2 = '0';      //Disable Group2 LEDS.
    }

    //When this function is run, the second 4 LEDS are disabled and the first if statement is enabled.

    if (Group1 == '1')
    {

        //Here you can turn on LEDS or something else.
    }

    else
    (

        //Here you can turn on LEDS or something else.
    )
}
/* END Constant Duty Cycle. */
/*
```

```

/*
** #####
**
** This file was created by Alex van den Biggelaar
** for the Prototype of the Primes Project Group 2011/2012.
** Linear Variable Duty Cycle
** #####
*/
int RefValue = 1; //This variable represents the variable reference value, start at one.
int cnt = 0; //This is the counter, start at 0.
boolean Check; //This variable is to determine addition or subtraction of the counter.
boolean PolCh1 = true; //This variable is to determine addition or subtraction of the RefVal, is
true.
int RefValue0 = 0; //This variable is to compare the result of RefVal.
int RefVal1 = 250; //This variable is to compare the result of RefVal.
int RefVal2 = 500; //This variable is to compare the result of RefVal.
int RefVal3 = 750; //This variable is to compare the result of RefVal.
int RefValue4 = 1000; //This variable is to compare the result of RefVal.

int main() //Start routine.
{

if(PolCh1 == true; Check == true) //If PolCh1 and Check are true.
{
    RefValue += 1; //RefValue plus one.
    cnt += 1; //Counter plus one.
}

if(PolCh1 == false; Check == true) //If PolCh1 is false and Check is true.
{
    RefValue -= 1; //RefValue minus one.
    cnt += 1; //Counter plus one.
}

if(PolCh1 == true; Check == false) //If PolCh1 is true and Check is false.
{
    RefValue += 1; //RefValue plus one.
    cnt -= 1; //Counter minus one.
}

if(PolCh1 == false; Check == false) //If PolCh1 and Check are false.
{
    RefValue -= 1; //RefValue minus one.
    cnt += 1; //Counter plus one.
}

if(cnt == 1000) //If counter equals 1000.
{
    Check = false; //Set Check to false, start counting counter backwards.
}

if(cnt == 0) //If counter equals 0.
{
    Check = true; //Set Check to true, start counting counter upwards.
}

if(RefValue == RefValue4) //If RefValue equals to RefValue4.
{
    PolCh1 = false; //Set PolCh1 to false, start counting RefValue downwards.
}

if(RefValue == RefValue0) //If RefValue equals to RefValue0.
{
    PolCh1 = true; //Set PolCh1 to true, start counting RefValue upwards.
}

//Here follow all if functions to compare the RefValue to the RefValue 0,1,2,3,4 following a specific signal.

if(PolCh1 == true; RefValue >= RefVal0)
{
    //signal1 on, all order signals off
}

if(PolCh1 == true; RefValue >= RefVal1)

```

```

{
    //signals 1,2 on, all order signals off
}

if(PolCh1 == true; RefValue >= RefVal2)
{
    //signals 1,2,3 on, all order signals off
}

if(PolCh1 == true; RefValue >= RefVal3)
{
    //signals 1,2,3,4 on, all order signals off
}

if(PolCh1 == false; RefValue <= RefVal4)
{
    //signal8 on, all order signals off
}

if(PolCh1 == false; RefValue <= RefVal3)
{
    //signals 7,8 on, all order signals off
}

if(PolCh1 == false; RefValue <= RefVal2)
{
    //signals 6,7,8 on, all order signals off
}

if(PolCh1 == false; RefValue <= RefVal1)
{
    //signals 5,6,7,8 on, all order signals off
}

}
/* END Linear Duty Cycle. */

/*
** #####
** THIS BEAN MODULE IS GENERATED BY THE TOOL. DO NOT MODIFY IT.
** Filename : Cpu.C
** Project : MC56F8006_LED_LAB
** Processor : MC56F8006_48_LQFP
** Beantype : 56F8006_48_LQFP
** Version : Bean 01.016, Driver 02.01, CPU db: 3.00.177
** Datasheet : MC56F8006RM,Rev. 0 Draft B,09/2007
** Compiler : Metrowerks DSP C Compiler
** Date/Time : 13/01/2002, 03:23
** Abstract :
**
** Settings :
**
** Contents :
** EnableInt - void Cpu_EnableInt(void);
** DisableInt - void Cpu_DisableInt(void);
** SetWaitMode - void Cpu_SetWaitMode(void);
** SetStopMode - void Cpu_SetStopMode(void);
** #####*/

/* MODULE Cpu. */
#include "LED1.h"
#include "Inhr6.h"
#include "LED2.h"
#include "Inhr5.h"
#include "LED3.h"
#include "Inhr4.h"
#include "LED4.h"
#include "Inhr3.h"
#include "LED5.h"
#include "Inhr2.h"
#include "LED6.h"
#include "Inhr1.h"
#include "T11.h"

```

```

#include "Bits1.h"
#include "Bits2.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "Events.h"
#include "Cpu.h"

/* Global variables */
volatile word SR_lock = 0;      /* Lock */
volatile word SR_reg;         /* Current value of the SR register */
/*
=====
** Method   : Cpu_Interrupt (bean 56F8006_48_LQFP)
**
** Description :
**   The method services unhandled interrupt vectors.
**   This method is internal. It is used by Processor Expert only.
** =====
*/
#pragma interrupt alignsp
void Cpu_Interrupt(void)
{
    asm(DEBUGHLT);           /* Halt the core and placing it in the debug processing state */
}

/*
=====
** Method   : Cpu_DisableInt (bean 56F8006_48_LQFP)
**
** Description :
**   Disables all maskable interrupts
** Parameters : None
** Returns   : Nothing
** =====
*/
/*
/*
void Cpu_DisableInt(void)

**   This method is implemented as macro in the header module. **
*/

/*
=====
** Method   : Cpu_EnableInt (bean 56F8006_48_LQFP)
**
** Description :
**   Enables all maskable interrupts
** Parameters : None
** Returns   : Nothing
** =====
*/
/*
/*
void Cpu_EnableInt(void)

**   This method is implemented as macro in the header module. **
*/

/*
=====
** Method   : Cpu_SetStopMode (bean 56F8006_48_LQFP)
**
** Description :
**   Sets low power mode - Stop mode.
**   For more information about the stop mode see this CPU
**   documentation.
** Parameters : None
** Returns   : Nothing
** =====
*/
/*
/*
void Cpu_SetStopMode(void)

**   This method is implemented as macro in the header module. **
*/
/*

```

```

** =====
** Method   : Cpu_SetWaitMode (bean 56F8006_48_LQFP)
**
** Description :
**   Sets low power mode - Wait mode.
**   For more information about the wait mode see this CPU
**   documentation.
**   Release from wait mode: Reset or interrupt
** Parameters : None
** Returns   : Nothing
** =====
*/
/*
void Cpu_SetWaitMode(void)

**   This method is implemented as macro in the header module. **
*/

/*
** =====
** Method   : _EntryPoint (bean 56F8006_48_LQFP)
**
** Description :
**   Initializes the whole system like timing and so on. At the end
**   of this function, the C startup is invoked to initialize stack,
**   memory areas and so on.
**   This method is internal. It is used by Processor Expert only.
** =====
*/
extern void init_56800_(void); /* Forward declaration of external startup function declared in startup file */

/** !!! Here you can place your own code using property "User data declarations" on the build options tab. !!! ***/

void _EntryPoint(void)
{
    #pragma constarray off

    /** !!! Here you can place your own code before PE initialization using property "User code before PE initialization" on
    the build options tab. !!! ***/

    /** ### MC56F8006_48_LQFP "Cpu" init code ... ***/
    /** PE initialization code after reset ***/
    /* System clock initialization */
    setRegBitGroup(OCCS_OCTRL, TRIM, (word)getReg(FM_OPT1) & 0x03FF); /* Set the trim osc freq with factory
    programmed value */
    setRegBit(OCCS_OCTRL, CLK_MODE); /* Select an internal oscillator mode */
    clrRegBit(OCCS_CTRL, PRECS); /* Select an internal clock source for the CPU core */
    setReg(OCCS_CTRL, (OCCS_CTRL_LCKON_MASK | OCCS_CTRL_ZSRC0_MASK)); /* Enable PLL, LCKON and
    select clock source from prescaler */
    /* OCCS_DIVBY: LORTP=2,COD=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0 */
    setReg16(OCCS_DIVBY, 0x2000); /* Set the clock prescalers */
    while(!getRegBit(OCCS_STAT, LCK0)){ /* Wait for PLL lock */
    setReg(OCCS_CTRL, (OCCS_CTRL_LCKON_MASK | OCCS_CTRL_ZSRC1_MASK)); /* Select clock source from
    postscaler */
    /* FM_CLKDIV: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,PRDIV8=0,DIVLD=0,PRDIV8=0,DIV=0x28 */
    setReg16(FM_CLKDIV, 0x28); /* Set the flash clock prescaler */
    /** End of PE initialization code after reset ***/

    /** !!! Here you can place your own code after PE initialization using property "User code after PE initialization" on the
    build options tab. !!! ***/

    setRegBit(SIM_PCE, COP); /* Enable COP peripheral clocks */
    setReg(COP_CTRL, 0); /* Disable COP running after reset */
    clrRegBit(SIM_PCE, COP); /* Disable COP peripheral clocks */
    asm(JMP init_56800_); /* Jump to C startup code */
}

/*
** =====
** Method   : PE_low_level_init (bean 56F8006_48_LQFP)
**
** Description :
**   Initializes beans and provides common register initialization.
**   The method is called automatically as a part of the
**   application initialization code.
**   This method is internal. It is used by Processor Expert only.
** =====
*/
void PE_low_level_init(void)

```

```

{
/*
GPIO_A_DRIVE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,DRIVE7=0,DRIVE6=0,DRIVE5=0,DRIVE4=0,DRIVE3=0,DRIVE2=0,DRIVE1=0,DRIVE0=0 */
setReg16(GPIO_A_DRIVE, 0x00); /* Set High/Low drive strength on GPIOA pins according to the CPU bean settings */
/*
GPIO_B_DRIVE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,DRIVE7=0,DRIVE6=0,DRIVE5=0,DRIVE4=0,DRIVE3=0,DRIVE2=0,DRIVE1=0,DRIVE0=0 */
setReg16(GPIO_B_DRIVE, 0x00); /* Set High/Low drive strength on GPIOB pins according to the CPU bean settings */
/*
GPIO_C_DRIVE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,DRIVE7=0,DRIVE6=0,DRIVE5=0,DRIVE4=0,DRIVE3=0,DRIVE2=0,DRIVE1=0,DRIVE0=0 */
setReg16(GPIO_C_DRIVE, 0x00); /* Set High/Low drive strength on GPIOC pins according to the CPU bean settings */
/*
GPIO_D_DRIVE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,DRIVE3=0,DRIVE2=0,DRIVE1=0,DRIVE0=0 */
setReg16(GPIO_D_DRIVE, 0x00); /* Set High/Low drive strength on GPIOD pins according to the CPU bean settings */
/*
GPIO_E_DRIVE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,DRIVE7=0,DRIVE6=0,DRIVE5=0,DRIVE4=0,DRIVE3=0,DRIVE2=0,DRIVE1=0,DRIVE0=0 */
setReg16(GPIO_E_DRIVE, 0x00); /* Set High/Low drive strength on GPIOE pins according to the CPU bean settings */
/*
GPIO_F_DRIVE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,DRIVE3=0,DRIVE2=0,DRIVE1=0,DRIVE0=0 */
setReg16(GPIO_F_DRIVE, 0x00); /* Set High/Low drive strength on GPIOF pins according to the CPU bean settings */
/*
GPIO_A_IFE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,IFE7=0,IFE6=0,IFE5=0,IFE4=0,IFE3=0,IFE2=0,IFE1=0,IFE0=0 */
setReg16(GPIO_A_IFE, 0x00); /* Set the input filter on GPIOA pins according to the CPU bean settings */
/*
GPIO_B_IFE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,IFE7=0,IFE6=0,IFE5=0,IFE4=0,IFE3=0,IFE2=0,IFE1=0,IFE0=0 */
setReg16(GPIO_B_IFE, 0x00); /* Set the input filter on GPIOB pins according to the CPU bean settings */
/*
GPIO_C_IFE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,IFE7=0,IFE6=0,IFE5=0,IFE4=0,IFE3=0,IFE2=0,IFE1=0,IFE0=0 */
setReg16(GPIO_C_IFE, 0x00); /* Set the input filter on GPIOC pins according to the CPU bean settings */
/* GPIO_D_IFE: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,IFE3=0,IFE2=0,IFE1=0,IFE0=0 */
setReg16(GPIO_D_IFE, 0x00); /* Set the input filter on GPIOD pins according to the CPU bean settings */
/*
GPIO_E_IFE:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,IFE7=0,IFE6=0,IFE5=0,IFE4=0,IFE3=0,IFE2=0,IFE1=0,IFE0=0 */
setReg16(GPIO_E_IFE, 0x00); /* Set the input filter on GPIOE pins according to the CPU bean settings */
/* GPIO_F_IFE: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,IFE3=0,IFE2=0,IFE1=0,IFE0=0 */
setReg16(GPIO_F_IFE, 0x00); /* Set the input filter on GPIOF pins according to the CPU bean settings */
/*
GPIO_A_SLEW:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,SLEW7=1,SLEW6=1,SLEW5=1,SLEW4=1,SLEW3=1,SLEW2=1,SLEW1=1,SLEW0=1 */
setReg16(GPIO_A_SLEW, 0xFF); /* Set the input filter on GPIOA pins according to the CPU bean settings */
/*
GPIO_B_SLEW:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,SLEW7=1,SLEW6=1,SLEW5=1,SLEW4=1,SLEW3=1,SLEW2=1,SLEW1=1,SLEW0=1 */
setReg16(GPIO_B_SLEW, 0xFF); /* Set the input filter on GPIOB pins according to the CPU bean settings */
/*
GPIO_C_SLEW:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,SLEW7=1,SLEW6=1,SLEW5=1,SLEW4=1,SLEW3=1,SLEW2=1,SLEW1=1,SLEW0=1 */
setReg16(GPIO_C_SLEW, 0xFF); /* Set the input filter on GPIOC pins according to the CPU bean settings */
/*
GPIO_D_SLEW:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,SLEW3=1,SLEW2=1,SLEW1=1,SLEW0=1 */
setReg16(GPIO_D_SLEW, 0x0F); /* Set the input filter on GPIOD pins according to the CPU bean settings */
/*
GPIO_E_SLEW:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,SLEW7=1,SLEW6=1,SLEW5=1,SLEW4=1,SLEW3=1,SLEW2=1,SLEW1=1,SLEW0=1 */
setReg16(GPIO_E_SLEW, 0xFF); /* Set the input filter on GPIOE pins according to the CPU bean settings */
/*
GPIO_F_SLEW:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,SLEW3=1,SLEW2=1,SLEW1=1,SLEW0=1 */
setReg16(GPIO_F_SLEW, 0x0F); /* Set the input filter on GPIOF pins according to the CPU bean settings */
/*
SIM_PCR:
TMR_CR=0,??=0,PWM_CR=0,SCI_CR=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0 */
setReg16(SIM_PCR, 0x00); /* Set the TMR; PWM; SCI module clock rates */
/*
SIM_PCE:
CMP2=0,CMP1=0,CMP0=0,ADC1=0,ADC0=0,PGA1=0,PGA0=0,I2C=0,SCI=0,SPI=0,PWM=0,COP=0,PDB=0,PIT=1,TAO=0 */
setReg16(SIM_PCE, 0x04); /* Set up the peripheral clock enable register */
/*
SIM_SDR:
CMP2=0,CMP1=0,CMP0=0,ADC1=0,ADC0=0,PGA1=0,PGA0=0,I2C=0,SCI=0,SPI=0,PWM=0,COP=0,PDB=0,PIT=0,TAO=0 */
setReg16(SIM_SDR, 0x00); /* Set up the STOP disable register */

```

```

/*                                                                 SIM_CTRL:
??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,ONCEEBL=0,SWRST=0,STOP_DISABLE=0,WAIT_DISABLE
=0 */
setReg16(SIM_CTRL, 0x00);      /* Set up the SIM control register */
/* SIM_CLKOUT: ??=0,??=0,CLKDIS1=1,??=0,??=0,CLKOSEL1=0,??=0,??=0,CLKDIS0=1,CLKOSEL0=0 */
setReg16(SIM_CLKOUT, 0x2020); /* Set up the SIM clock output select register */
/*                                                                 PMC_SCR:
OORF=0,LVDF=0,PPDF=0,PORF=0,OORIE=0,LVDIE=0,LVDRE=1,PPDE=0,LPR=0,LPRS=0,LPWUI=0,BGBE=0,LVD
E=0,LVLS=0,PROT=0 */
setReg16(PMC_SCR, 0x0200);
/* Common initialization of the CPU registers */
/* GPIO_A_IENR: IEN5=0,IEN4=0,IEN3=0,IEN2=0,IEN1=0,IEN0=0 */
clrReg16Bits(GPIO_A_IENR, 0x3F);
/* GPIO_A_IESR: IES5=1,IES4=1,IES3=1,IES2=1,IES1=1,IES0=1 */
setReg16Bits(GPIO_A_IESR, 0x3F);
/* GPIO_A_DR: D5=0,D4=0,D3=0,D2=0,D1=0,D0=0 */
clrReg16Bits(GPIO_A_DR, 0x3F);
/* GPIO_A_DDR: DD5=1,DD4=1,DD3=1,DD2=1,DD1=1,DD0=1 */
setReg16Bits(GPIO_A_DDR, 0x3F);
/* GPIO_A_PER: PE5=0,PE4=0,PE3=0,PE2=0,PE1=0,PE0=0 */
clrReg16Bits(GPIO_A_PER, 0x3F);
/* GPIO_E_IENR: IEN7=0,IEN6=0,IEN1=0,IEN0=0 */
clrReg16Bits(GPIO_E_IENR, 0xC3);
/* GPIO_E_IESR: IES7=1,IES6=1,IES1=1,IES0=1 */
setReg16Bits(GPIO_E_IESR, 0xC3);
/* GPIO_E_DR: D7=0,D6=0,D1=0,D0=0 */
clrReg16Bits(GPIO_E_DR, 0xC3);
/* GPIO_E_DDR: DD7=1,DD6=1,DD1=1,DD0=1 */
setReg16Bits(GPIO_E_DDR, 0xC3);
/* GPIO_E_PER: PE7=0,PE6=0,PE1=0,PE0=0 */
clrReg16Bits(GPIO_E_PER, 0xC3);
/* ### TimerInt "TI1" init code ... */
TI1_Init();
__EI(0);          /* Enable interrupts of the selected priority level */
}

/* END Cpu. */

/*

/*
** #####
**
** This file was created by Alex van den Biggelaar
** for the Prototype of the Primes Project Group 2011/2012.
** Main.C
**
** #####
** */

/** #####
** Filename : MC56F8006_LED_LAB.C
** Project  : MC56F8006_LED_LAB
** Processor : MC56F8006_48_LQFP
** Version  : Driver 01.13
** Compiler  : Metrowerks DSP C Compiler
** Date/Time : 1/12/2009, 1:28 PM
** Abstract  :
**   Main module.
**   This module contains user's application code.
** Settings  :
** Contents  :
**   No public methods
**
** #####*/
/* MODULE MC56F8006_LED_LAB */

/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "LED1.h"
#include "Inhr6.h"
#include "LED2.h"
#include "Inhr5.h"
#include "LED3.h"
#include "Inhr4.h"
#include "LED4.h"

```

```

#include "Inhr3.h"
#include "LED5.h"
#include "Inhr2.h"
#include "LED6.h"
#include "Inhr1.h"
#include "T11.h"
#include "Bits1.h"
#include "Bits2.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

void main(void)
{
    /* Write your local variable definition here */

    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
    PE_low_level_init();
    /*** End of Processor Expert internal initialization.      ***/

    /* Write your code here */

}

/* END Main. */

/*

/** #####
** THIS BEAN MODULE IS GENERATED BY THE TOOL. DO NOT MODIFY IT.
** Filename : T11.C
** Project : MC56F8006_LED_LAB
** Processor : MC56F8006_48_LQFP
** Beantype : TimerInt
** Version : Bean 02.157, Driver 02.00, CPU db: 3.00.177
** Compiler : Metrowerks DSP C Compiler
** Date/Time : 13/01/2002, 03:10
** Abstract :
** This bean "TimerInt" implements a periodic interrupt.
** When the bean and its events are enabled, the "OnInterrupt"
** event is called periodically with the period that you specify.
** TimerInt supports also changing the period in runtime.
** The source of periodic interrupt can be timer compare or reload
** register or timer-overflow interrupt (of free running counter).
** Settings :
** Timer name : PIT (16-bit)
** Compare name : PIT_Modulo
** Counter shared : No
**
** High speed mode
** Prescaler : divide-by-1
** Clock : 32000000 Hz
** Initial period/frequency
** Xtal ticks : 80
** microseconds : 10
** seconds (real) : 0.00001
** Hz : 100000
** kHz : 100
**
** Runtime setting : none
**
** Initialization:
** Timer : Enabled
** Events : Enabled
**
** Timer registers
** Counter : PIT_CNTR [F2E2]
** Mode : PIT_CTRL [F2E0]
** Run : PIT_CTRL [F2E0]

```



```

**      Prescaler      : PIT_CTRL [F2E0]
**
**      Compare registers
**      Compare       : PIT_MOD [F2E1]
**
**      Flip-flop registers
**      Contents :
**      No public methods
**
** #####*/

/* MODULE T11. */

#include "Events.h"
#include "T11.h"

/* Internal method prototypes */
static void SetCV(word Val);
static void SetPV(byte Val);

/*
** =====
** Method   : SetCV (bean TimerInt)
**
** Description :
**   Sets compare or preload register value. The method is called
**   automatically as a part of several internal methods.
**   This method is internal. It is used by Processor Expert only.
** =====
*/
static void SetCV(word Val)
{
    setReg(PIT_MOD,Val);      /* Store given value to the compare register */
}

/*
** =====
** Method   : SetPV (bean TimerInt)
**
** Description :
**   Sets prescaler value. The method is called automatically as a
**   part of several internal methods.
**   This method is internal. It is used by Processor Expert only.
** =====
*/
static void SetPV(byte Val)
{
    setRegBitGroup(PIT_CTRL,PRESCALER,Val); /* Store given value to the prescaler */
}

/*
** =====
** Method   : T11_Init (bean TimerInt)
**
** Description :
**   Initializes the associated peripheral(s) and the beans
**   internal variables. The method is called automatically as a
**   part of the application initialization code.
**   This method is internal. It is used by Processor Expert only.
** =====
*/
void T11_Init(void)
{
    /* PIT_CTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,PRESCALER=0,PRF=0,PRIE=1,CNT_EN=0 */
    setReg(PIT_CTRL,0x02);      /* Set up control register */
    SetCV((word)0x0140);      /* Store appropriate value to the compare register according to the selected high
speed CPU mode */
    SetPV((byte)0x00);      /* Set prescaler register according to the selected high speed CPU mode */
    setRegBit(PIT_CTRL,CNT_EN); /* Run counter */
}

/*
** =====
** Method   : T11_Interrupt (bean TimerInt)
**
** Description :
**   The method services the interrupt of the selected peripheral(s)

```

```

**      and eventually invokes the beans event(s).
**      This method is internal. It is used by Processor Expert only.
** =====
*/
#pragma interrupt alignsp saveall
void T11_Interrupt(void)
{
  clrRegBit(PIT_CTRL,PRF);      /* Reset interrupt request flag */
  T11_OnInterrupt();           /* Invoke user event */
}

/* END T11. */

/*
** #####
** This file was created by Alex van den Biggelaar
** for the Prototype of the Primes Project Group 2011/2012.
** Bits1.C
** #####
*/

/** #####
** THIS BEAN MODULE IS GENERATED BY THE TOOL. DO NOT MODIFY IT.
** Filename : Bits1.C
** Project  : MC56F8006_LED_LAB
** Processor : MC56F8006_48_LQFP
** Beantype  : BitsIO
** Version  : Bean 02.098, Driver 01.19, CPU db: 3.00.177
** Compiler : Metrowerks DSP C Compiler
** Date/Time : 13/01/2002, 03:23
** Abstract :
** This bean "BitsIO" implements a multi-bit input/output.
** It uses selected pins of one 1-bit to 8-bit port.
** Note: This bean is set to work in Output direction only.
** Settings :
** Port name          : GPIOE
**
** Bit mask of the port : 0003
** Number of bits/pins  : 2
** Single bit numbers  : 0 to 1
** Values range        : 0 to 3
**
** Initial direction   : Output (direction cannot be changed)
** Initial output value : 0 = 000H
** Initial pull option  : off
**
** Port data register  : GPIO_E_DR [F201]
** Port control register : GPIO_E_DDR [F202]
** Port function register : GPIO_E_PER [F203]
**
** -----
**      Bit | Pin | Name
** -----
**      0  |  5  | GPIOE0
**      1  |  6  | GPIOE1_ANB9_CMP0_P1
** -----
**
** Optimization for      : speed
** Contents :
** GetDir - bool Bits1_GetDir(void);
** GetVal - byte Bits1_GetVal(void);
** PutVal - void Bits1_PutVal(byte Val);
** GetBit - bool Bits1_GetBit(byte Bit);
** PutBit - void Bits1_PutBit(byte Bit, bool Val);
** SetBit - void Bits1_SetBit(byte Bit);
** ClrBit - void Bits1_ClrBit(byte Bit);
** NegBit - void Bits1_NegBit(byte Bit);
**
** (c) Copyright UNIS, a.s. 1997-2008
** UNIS, a.s.
** Jundrovska 33
** 624 00 Brno
** Czech Republic
** http   : www.processorexpert.com
** mail   : info@processorexpert.com
** #####

```

```

/* MODULE Bits1. */

#include "Bits1.h"

/*Include shared modules, which are used for whole project*/
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

#include "Cpu.h"

static const byte Bits1_Table[2]={
0x01,0x02};          /* Table of mask constants */
/*
** =====
** Method   : Bits1_GetMsk (bean BitsIO)
**
** Description :
** Returns pin mask. The method is called automatically as a part
** of bit method.
** This method is internal. It is used by Processor Expert only.
** =====
*/
static byte Bits1_GetMsk(byte Value)
{
return((byte)((Value<0x02)?Bits1_Table[Value]:0x00)); /* Return appropriate bit mask */
}

/*
** =====
** Method   : Bits1_GetVal (bean BitsIO)
**
** Description :
** This method returns an input value.
** a) direction = Input : reads the input value from the
** pins and returns it
** b) direction = Output : returns the last written value
** Note: This bean is set to work in Output direction only.
** Parameters : None
** Returns    :
** ---      - Input value (0 to 3)
** =====
*/
/*
byte Bits1_GetVal(void)

** This method is implemented as a macro. See Bits1.h file. **
*/

/*
** =====
** Method   : Bits1_PutVal (bean BitsIO)
**
** Description :
** This method writes the new output value.
** Parameters :
** NAME      - DESCRIPTION
** Val       - Output value (0 to 3)
** Returns   : Nothing
** =====
*/
void Bits1_PutVal(byte Val)
{
setReg(GPIO_E_DR,((getReg(GPIO_E_DR)) & ~Bits1_PIN_MASK) | ((word)Val & Bits1_PIN_MASK)); /* Set-up bits
on port */
}

```

```

/*
** =====
** Method   : Bits1_GetBit (bean BitsIO)
**
** Description :
** This method returns the specified bit of the input value.
** a) direction = Input : reads the input value from pins
**    and returns the specified bit
** b) direction = Output : returns the specified bit
**    of the last written value
** Note: This bean is set to work in Output direction only.
** Parameters :
** NAME      - DESCRIPTION
** Bit       - Number of the bit to read (0 to 1)
** Returns   :
** ---      - Value of the specified bit (FALSE or TRUE)
**          FALSE = "0" or "Low", TRUE = "1" or "High"
** =====
*/
bool Bits1_GetBit(byte Bit)
{
    register byte Mask=Bits1_GetMsk(Bit); /* Temporary variable - bit mask */

    return((bool)((Mask)?((getReg(GPIO_E_DR)) & Mask) == Mask:0x00)); /* Return input value */
}

/*
** =====
** Method   : Bits1_PutBit (bean BitsIO)
**
** Description :
** This method writes the new value to the specified bit
** of the output value.
** Parameters :
** NAME      - DESCRIPTION
** Bit       - Number of the bit (0 to 1)
** Val       - New value of the bit (FALSE or TRUE)
**          FALSE = "0" or "Low", TRUE = "1" or "High"
** Returns   : Nothing
** =====
*/
void Bits1_PutBit(byte Bit, bool Val)
{
    register byte Mask=Bits1_GetMsk(Bit); /* Temporary variable - bit mask */

    if (Mask) /* Is bit mask correct? */
        if (Val) { /* Is it one to be written? */
            setRegBits(GPIO_E_DR,Mask); /* Set appropriate bit on port */
        }
        else { /* Is it zero to be written? */
            clrRegBits(GPIO_E_DR,Mask); /* Clear appropriate bit on port */
        }
}

/*
** =====
** Method   : Bits1_SetBit (bean BitsIO)
**
** Description :
** This method sets (sets to one) the specified bit of the
** output value.
** [ It is the same as "PutBit(Bit,TRUE);" ]
** Parameters :
** NAME      - DESCRIPTION
** Bit       - Number of the bit to set (0 to 1)
** Returns   : Nothing
** =====
*/
void Bits1_SetBit(byte Bit)
{
    register byte Mask=Bits1_GetMsk(Bit); /* Temporary variable - bit mask */

    if (Mask) { /* Is bit mask correct? */
        setRegBits(GPIO_E_DR,Mask); /* Set appropriate bit on port */
    }
}

/*
** =====

```

```

** Method   : Bits1_ClrBit (bean BitsIO)
**
** Description :
**   This method clears (sets to zero) the specified bit
**   of the output value.
**   [ It is the same as "PutBit(Bit, FALSE);" ]
** Parameters :
**   NAME      - DESCRIPTION
**   Bit       - Number of the bit to clear (0 to 1)
** Returns    : Nothing
** =====
*/
void Bits1_ClrBit(byte Bit)
{
    register byte Mask=Bits1_GetMsk(Bit); /* Temporary variable - bit mask */

    if (Mask) { /* Is bit mask correct? */
        clrRegBits(GPIO_E_DR, Mask); /* Clear appropriate bit on port */
    }
}

/*
** =====
** Method   : Bits1_NegBit (bean BitsIO)
**
** Description :
**   This method negates (inverts) the specified bit of the
**   output value.
** Parameters :
**   NAME      - DESCRIPTION
**   Bit       - Number of the bit to invert (0 to 31)
** Returns    : Nothing
** =====
*/
void Bits1_NegBit(byte Bit)
{
    register byte Mask=Bits1_GetMsk(Bit); /* Temporary variable - bit mask */

    if (Mask) { /* Is bit mask correct? */
        changeRegBits(GPIO_E_DR, Mask); /* Negate appropriate bit on port */
    }
}

/*
** =====
** Method   : Bits1_GetDir (bean BitsIO)
**
** Description :
**   This method returns direction of the bean.
** Parameters : None
** Returns    :
**   ---      - Direction of the bean (always TRUE, Output only)
**   FALSE = Input, TRUE = Output
** =====
*/
bool Bits1_GetDir(void)

** This method is implemented as a macro. See Bits1.h file. **
*/

/* END Bits1. */

```

```

/*
** #####
** This file was created by Alex van den Biggelaar
** for the Prototype of the Primes Project Group 2011/2012.
** #####
*/

/* #####
** Filename : Events.C
** Project : MC56F8006_LED_LAB
** Processor : MC56F8006_48_LQFP
** Beantype : Events
** Version : Driver 01.03
** Compiler : Metrowerks DSP C Compiler
** Date/Time : 1/12/2009, 1:28 PM
** Abstract :
** This is user's event module.
** Put your event handler code here.
** Settings :
** Contents :
** Tl1_OnInterrupt - void Tl1_OnInterrupt(void);
** #####
*/ MODULE Events */

#include "Cpu.h"
#include "Events.h"

int count = 0 ; // use this count to determine what bits to toggle;
                // increment the count and do the toggle at the isr
                // of the timer.
int index = 0;

/*
** =====
** Event : Tl1_OnInterrupt (module Events)
**
** From bean : Tl1 [TimerInt]
** Description :
** When a timer interrupt occurs this event is called (only
** when the bean is enabled - <Enable> and the events are
** enabled - <EnableEvent>). This event is enabled only if a
** <interrupt service/event> is enabled.
** Parameters : None
** Returns : Nothing
** =====
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
/* is set to 'yes' (#pragma interrupt saveall is generated before the ISR) */
void Tl1_OnInterrupt(void)
{

while(index <= 417)
{
    if(index<125 )
    {
        Bits1_PutBit(0, 0);
        Bits1_PutBit(1, 0);
        Bits2_PutBit(0, 1);
        Bits2_PutBit(1, 1);

        index++;
    }
    else if(index>=104 && index<=208 )
    {
        Bits1_PutBit(0, 1);
        Bits1_PutBit(1, 1);

        Bits2_PutBit(0, 0);
        Bits2_PutBit(1, 0);

        index++;
    }
    else if(index>=289 && index<=312 )
    {
        Bits1_PutBit(0, 0);

```

```

        Bits1_PutBit(1, 0);

        Bits2_PutBit(0, 1);
        Bits2_PutBit(1, 1);

        index++;
    }
    else if(index>=313 && index<417 )
    {
        Bits1_PutBit(0, 1);
        Bits1_PutBit(1, 1);

        Bits2_PutBit(0, 0);
        Bits2_PutBit(1, 0);

        index++;
    }
    else
    {
        index = 0;
    }
}

}

*/
** =====
** Event   : Button_OnInterrupt (module Events)
**
** From bean : Button [ExtInt]
** Description :
**   This event is called when an active signal edge/level has
**   occurred.
** Parameters : None
** Returns   : Nothing
** =====
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
/* is set to 'yes' (#pragma interrupt saveall is generated before the ISR) */
void Button_OnInterrupt(void)
{
}

    int ch = 0;

    if(ch == 0)
    {
        RefVal++;
    }

    if(ch == 1)
    {
        RefVal--;
    }

    if(RefVal == Min && ch == 1)
    {
        ch = 0;
    }

    if(RefVal == Max && ch == 0)
    {
        ch = 1;
    }
    /* place your Button interrupt procedure body here

    switch()
    {
        case 1:
            {
                setHz();
            }
            break;

```

```

        case 2:
            {
                setHz();
            }
            break;

```

```

/*
** =====
** Event   : Button2_OnInterrupt (module Events)
**
** From bean : Button2 [ExtInt]
** Description :
**   This event is called when an active signal edge/level has
**   occurred.
** Parameters : None
** Returns   : Nothing
** =====
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers' property */
/* is set to 'yes' (#pragma interrupt saveall is generated before the ISR) */
void Button2_OnInterrupt(void)
{
    BitsG1_ClrBit(0);
    BitsG1_ClrBit(1);
    BitsG1_ClrBit(2);
    BitsG1_ClrBit(3);
    BitsG2_ClrBit(4);
    BitsG2_ClrBit(5);
    BitsG2_ClrBit(6);
    BitsG2_ClrBit(7);
    LED6_Set(0);
    LED5_Set(0);
    LED4_Set(0);
    LED3_Set(0);
    LED2_Set(0);
    LED1_Set(0);
    T11_Disable();
    LED1_Off();
    LED2_Off();
    LED3_Off();
    LED4_Off();
    LED5_Off();
    LED6_Off();
    /* place your Button2 interrupt procedure body here */
}

/* END Events */

```

On the included CD, Getting Started CD Freescale semiconductor, you will find all the information of the Digital Signal Controller. That being: training, documentation and more software.

