CHALMERS

IMPLEMENTATION OF THE DIGITAL CONTROL BLOCK IN A LOW-POWER, LOW-COST RFID TAG

By

Amin Roostaei and Maziyar Yousefi Namini

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Integrated Electronic System Design
Chalmers University of Technology
August 2010

The Authors grant to Chalmers University of Technology and University of Gothenburg the nonexclusive Right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Authors warrant that they are the authors to the Work, and warrant that the Work does not contain text, pictures or other material that violates copyright law.

The Authors shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Authors have signed a copyright agreement with a third party regarding the Work, the Authors warrant hereby that they have obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Implementation of the Digital Control Block in a Low-Power, Low-Cost RFID Tag

Amin Roostaei,
Maziyar Yousefi Namini,
© Amin Roostaei, August 2010.
© Maziyar Yousefi Namini, August 2010.
Examiner: Lars Bengtsson
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden August 2010

ABSTRACT

Implementation of the Digital Control Block in a Low-Power, Low-Cost RFID Tag

By Amin Roostaei and Maziyar Yousefi Namini

Chairperson of the Supervisory Committee:
Lars Bengtsson
Department of Computer engineering

A thesis presented on the Designing of the power efficient and low cost RFID TAG with implementation of EGON protocol. The EGON protocol is the new communication protocol for RFID which was invented in Lepton Radio Co. in Sweden.

This thesis is based on three phases. In the first phase the VHDL code is written and tested with the ModelSim tool and in the next step, it is implemented on an FPGA and Verified. The last phase is mainly focused on the ASIC design.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The authors wish to express their appreciation to Professor Lars Bengtsson for his assistance and support during the course and development of this thesis. In addition, special thanks go to Lennart Hansson for his cooperation in providing the required hardware and Tung Hoang for his advice regarding the backend phase.

Finally, we offer our regards to Mohammad Attari, Ashkan Gheysvandi and all of those who supported us in any respect for completing this project.

GLOSSARY

EDA: Electronic Design Automation

SPI: Serial peripheral interface

FIFO: First in First out

FPGA: Field Programmable Gate Array

LSB: Least Significant Bit

MSB: Most Significant Bit

RFID: Radio Frequency Identification

VHDL: Very high speed integrated circuit Hardware Description Language

SMWOT: State Machine without Temperature

TPT: Temperature Protocol Tester

TVGEN: Test Vector Generator

PWMT: Pulse width Modulator Tester

MFTVG: Manual Final Test Vector Generator

AFTVG: Auto. Final Test Vector Generator

FSL: Fast Simplex Link

I N T R O D U C T I O N

RFID simply stands for Radio Frequency Identification.
As the name suggests, it is useful for identifying objects from a long distance. There are generally three types of RFID TAGs.
Active tags, which contain a battery and communicate with the reader from long distances.
Passive tags, which do not contain any power source in the tag, and they require an external source to transmit the signal, and finally Battery Assisted Passive (BAP) tags, which do not have an internal battery, but they have a higher range compared to the passive tags[7].
All RFIDs can read one TAG at the same time, but in this project an active RFID tag has been developed that the reader, is capable of reading several tags at the same time.
This protocol that is formally known as the EGON protocol was invented at Lepton Radio AB. in Sweden.
This project has been designed according to the EGON protocol reference, which provided by Lepton Radio AB with some minor changes.
In the original system there was no protocol to allow the transmission of the temperature. After investigation of different possibilities, the temperature transmission protocol was added to the EGON protocol. The full description is accessible in chapter 1 of this report.

Chapter 1

THE EGON PROTOCOL

The protocol uses 5 different frequencies to communicate. The frequencies are F0, F1, F2, F3, and F4 which are used by server (Reader) and Client (tags) to read specific data. Data can be ID or Temperature or both simultaneously. F0 is called beacon frequency which is sent by reader to wake up Tags. The other frequencies are used for two consecutive bits as F1: 00, F2: 01, F3: 10, F4: 11. The frequency of F0 is equal to 2.4+n0 GHZ and F1, F2, F3 and F4 is equal to 2.4 GHz + n0+n1, F1+n2, F2+n3, F3+n4 respectively where n0-n4 are chosen according to 2.45 GHZ ISM band. For more details refer to figure 1 [1].

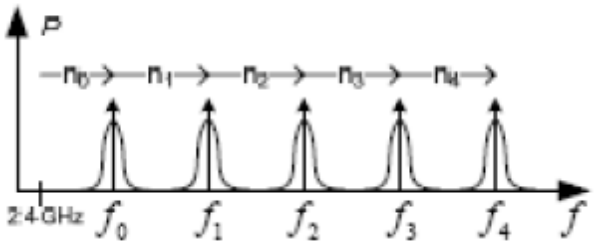

Figure 1. Frequency spectrum allocation [1]

The ID of the tag is read by reader. The reader sends F0 (Beacon Frequency) and awaking all tags in the region. Then the reader sends F1 to F4 frequencies and tags that have these frequencies on two first bits of ID respond and tune to them. At this point reader chooses a frequency and sends it back. Other tags with a different tuned frequency wait for the beacon frequency. Then, the reader sends other bits and Tag answers to them.

After sending ID, the tag goes to long sleep. For reading the temperature of tag after sending ID, the Reader sends F2 and if Tag responds back with F2, there is temperature to read. Then reader sends a Dummy F2 frequency and Tag sends the First bit. The reader sends F2 as acknowledgment of the previous bit consequently and Tag send the new bit and if there is problem in receiving temperature bits , the reader sends a frequency other than F2 and F0 as negative acknowledge (Nack) and Tag sends the corrupted bit again. When tag sends the last bit, the reader sends an Ack frequency and Tag responds with F2. The EGON Protocol is demonstrated in the following figures.

		Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10	Step11
		Reader TX Beacon	Reader TX F1-F4	Reader TX F1	Reader TX F3	Reader TX F4,F2	Reader TX F1					
No TEMP							No TEMP					
Tag 1	0010	Awakes	Respond F3	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon
Tag 2	1001	Awakes	Respond F2	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon
Tag 3	0100	Awakes	Respond F1	Respond F1	Respond F3	Respond F2	Long Sleep	Long Sleep	Long Sleep	Long Sleep	Long Sleep	Long Sleep
Tag 4	1100	Awakes	Respond F1	Respond F1	Respond F3	Respond F4	Long Sleep	Long Sleep	Long Sleep	Long Sleep	Long Sleep	Long Sleep

MSB Bit on Left (No Temperature)

Figure 2. EGON protocol without temperature

		Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10	Step11
		Reader TX Beacon	Reader TX F1-F4	Reader TX F1	Reader TX F3	Reader TX F4	Reader TX F2	Reader TX F2	Reader TX F1	Reader TX F2	Reader TX F2	
Temp =010							Start Sending Temperature	First Bit of Temp	Error on First Bit , send it again	Send second bit	Finish Sending	
Tag 1	0000	Awakes	Respond F1	Respond F1	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon
Tag 2	1000	Awakes	Respond F1	Respond F1	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon
Tag 3	1010	Awakes	Respond F3	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon	Await Beacon
Tag 4	1100	Awakes	Respond F1	Respond F1	Respond F3	Respond F4	Respond F2	Respond F3	Respond F3	Respond F2	Respond F2	Long Sleep

MSB Bit on Left (Temperature=010)

Figure 3. EGON protocol with temperature

Chapter 2

THE VHDL CODE DESCRIPTION

The main codes of the Tag consist of the My_data_type, Machine_state, PWM_Generator, SPI_MASTER_2, SPI_SLAVE, My_Components and General1 Files. In addition, there are other codes that are used for Test and Verification. The code structures and responsibility of each part are described in the following sections.

2.1 State machine and My_data_type

My_data_type file contains the definition of data type frequency which is used in different parts of the program.

State machine consists of 17 input ports and 2 output ports. Major parts of the input ports are specified for the ID, Temperature value (from sensor), the defined frequencies' value to response detected frequency, a pin to identify that a new frequency is detected and maximum sleep time after finishing operation. There are two output ports which send a value to other parts of the tag.

The code consists of five different states namely sending, tuning, await_beacon, Send_temp and long_sleep. Upon detection of input frequency (Osc_detect) and rising edge of the new detection frequency port (New data), the previous state will continue or new state is defined. In the await_beacon state, the tag waits for F0 frequency. Upon receiving F0 frequency, it goes to Tuning state. For solving the echo problem of F0, it is important to set F0 frequency's value to zero. In the tuning state, if the tag receives two least significant bits, it goes to sending state otherwise it goes to await_beacon state. In the sending state, the ID is checked and the state machine goes to the next state. If ID is failed it goes back to await_beacon state. In the Send_temp state after sending the temperature, it goes to long_sleep and waits for specific time, then goes back to await_beacon state.

2.1.1 Description issues

After writing and verifying the code in ModelSim and implementing it in FPGA, it works properly but in synthesizing step in cadence, only 24 test vectors passed. So after investigation of the problem, it was observed that sending state did not interpret properly in the Verilog file. The following code is problematic version of the code:

```
If (ID (cnt_ID) ='0' and ID (cnt_ID+1) ='0' and Osc_detect=F1) then
V1 <= V1_VALUE_F1;
V2 <= V2_VALUE_F1;
cnt_ID:=cnt_ID+1;
Elsif (ID (cnt_ID) ='1' and ID (cnt_ID+1) ='0' and Osc_detect=F2) then
V1 <= V1_VALUE_F2;
V2 <= V2_VALUE_F2;
cnt_ID:=cnt_ID+1;
Elsif (ID (cnt_ID) ='0' and ID (cnt_ID+1) ='1' and Osc_detect=F3) then
V1 <= V1_VALUE_F3;
V2 <= V2_VALUE_F3;
cnt_ID:=cnt_ID+1;
Elsif (ID (cnt_ID) ='1' and ID (cnt_ID+1) ='1' and Osc_detect=F4) then
V1 <= V1_VALUE_F4;
V2 <= V2_VALUE_F4;
cnt_ID:=cnt_ID+1;
Elsif (Osc_detect=F0) then
pr_state<= sending;
Else
pr_state<=await_beacon;
cnt_ID:=0;
End if;
```

This problem has been solved by the following code. In this code two least significant bits of ID have been shifted to a variable then it is checked.

```
ID_temp:=ID (cnt_ID+1 downto cnt_ID);
If (ID_temp (0) ='0' and ID_temp (1) ='0' and Osc_detect=F1) then
V1 <= V1_VALUE_F1;
V2 <= V2_VALUE_F1;
cnt_ID:=cnt_ID+1;
Elsif (ID_temp (0) ='1' and ID_temp (1) ='0' and Osc_detect=F2) then
```

… Continue the same procedure

Consequently, it is obvious that the cadence tool cannot synthesize the array which has index of variable definition.

2.2 PWM_Generator

It consists of five input ports and two output ports. Two input ports from state machine and one input port for new frequency detection. The outputs are two bits that are generated PWM output of the tag and goes to generator circuit to generate proper frequency.
The structure of the code is simple. It has a function to convert std_logic type to integer type and according to input from state machine, the output goes to high. It should be noticed that the resolution of output is 256 cycles and according to input from state machine number of high cycles is determined.

2.3 SPI_MASTER_2

It consists of three outputs and three inputs. In addition to Clock and reset, the SDI port is data input pin from SPI slave device. The SCK and SS pins are outputs clock to Slave Device and Enable pin respectively.
The SPI protocol (Serial Peripheral Interface) is a simple and risk free serial communication between different devices which is introduced by Motorola. Each device should have four pins for data input, enable, data output and clock. It has four modes of operation but mode zero is used in this project which means, data is captured on the clock's rising edge (low to high transition) and data is sent on a falling edge [2].

The code has implemented SPI master according to specification. The main clock is divided by ten and The SPI clock is generated. In the first step, SS goes low and a conversation is started. In the rising edge of SPI Clock data is read and stored. After each 8 bits, the clock is stopped for 2 clock cycles and a delay is generated between two bytes according to the SPI specification [2].The temperature is 16 bits, so after each 16 bits the procedure is repeated. The following schematic shows the SPI protocol.

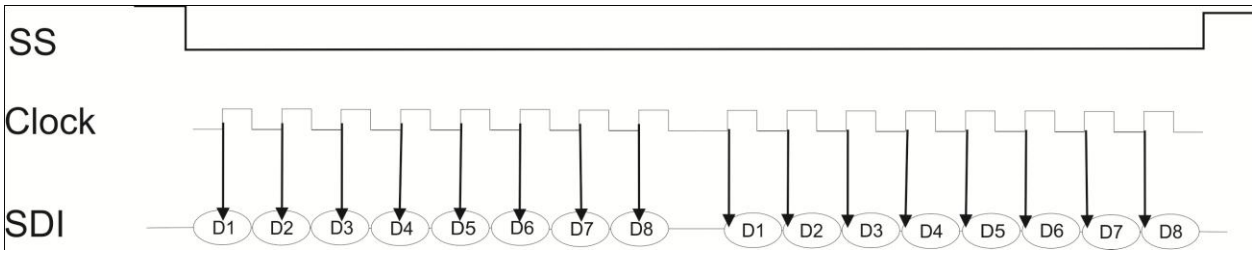

Figure 4. SPI Communication

2.4 SPI_SLAVE

It has twelve outputs which consist of Specific values for different frequencies and maximum time of sleep and ID. Beside that it has five inputs like SPI's Clock and SPI's enable and SPI's data.

If SS enable pin goes low, the communication is started and on the rising edge of the SPI's clock, the data is captured from SPI's data pin and stored in register. The whole required data is 136 bits, after receiving all data, data flows to output.

Note: It should be noted that SPI_Slave and SPI_MASTER are written based on the data sheet of two extern Device and sensor, U2C-12 USB[3] and LM74[6].

2.5 My_Components, General1

In My_Components section of the code, the component is defined to use as package in the other part of the code.

In the General1, the different part of the code is connected together as shown in figure 5. It should be mentioned that the final tag has eight input pins and four output pins. There are four input pins for SPI connection and 2 input pins for oscillation detection. Finally, two output ports for SPI and two output ports for Generated PWM are connected to the output pins of the tag.

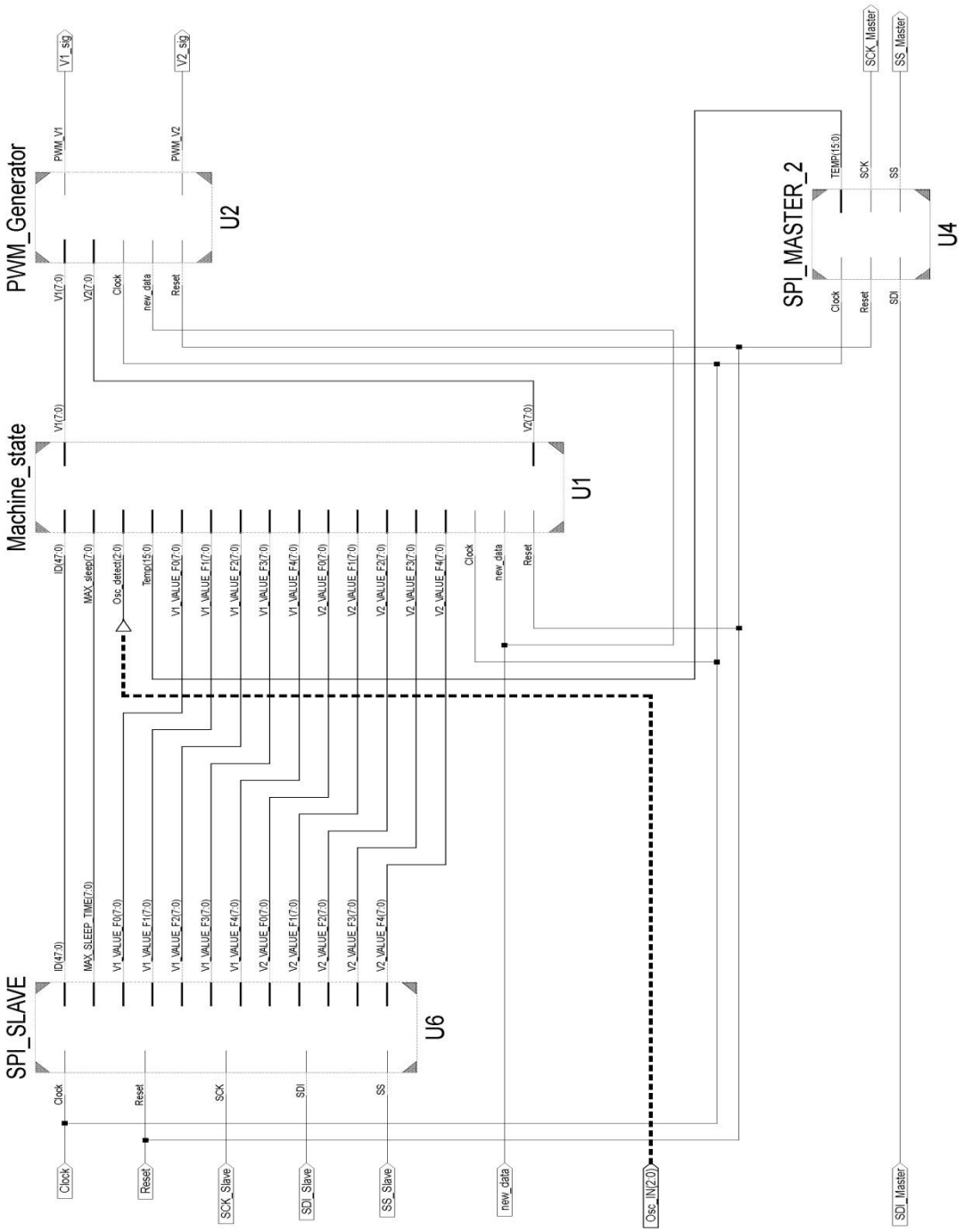

Figure 5. Schematic of TAG

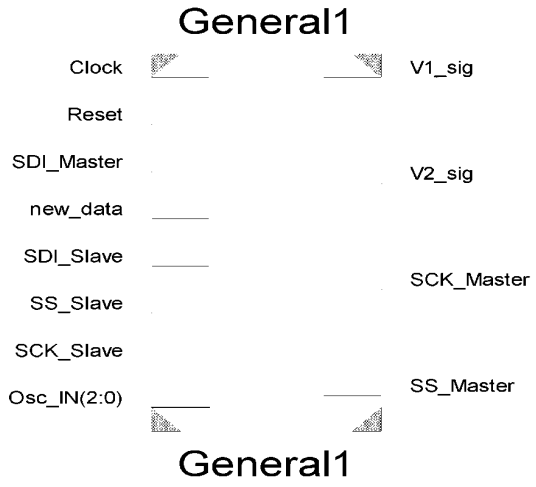

Figure 6. General1 Input/output Pins

2.6 SPI_Slave_FSL_Test

It is responsible for demonstration of Master SPI for SPI_SLAVE module. This module acts like SPI master and generate SPI clock and send data to FSL_In_Data port that has 136 bits including ID, Frequencies' value and sleep time. The Output data is sent out to SPI_SLAVE module. It should be noted that the above module is used only for test purpose.

2.7 SPI_Master_FSL_Test

It is responsible for demonstration of SPI Slave for SPI_master module. This module simulates the LM75 temperature sensor. The new temperature is fed through input port, and when the module receives clock and enable from SPI_MASTER, data is sentout.

2.8 Test_FSL_PWM

The code is communicated with PWM_Generator module and receives PWM of tag and converts it to a readable digit in order to test the tag easily. The method of code implementation is simple. It counts the number of cycles that PWM input ports are at high level. In addition, after completing operation, an output signal is raised and indicates that the operation is completed.

2.9 General

The module is for attaching the whole modules together. The tag module and the test modules are connected and make it ready to for test. The connection between components is shown in the following figure.

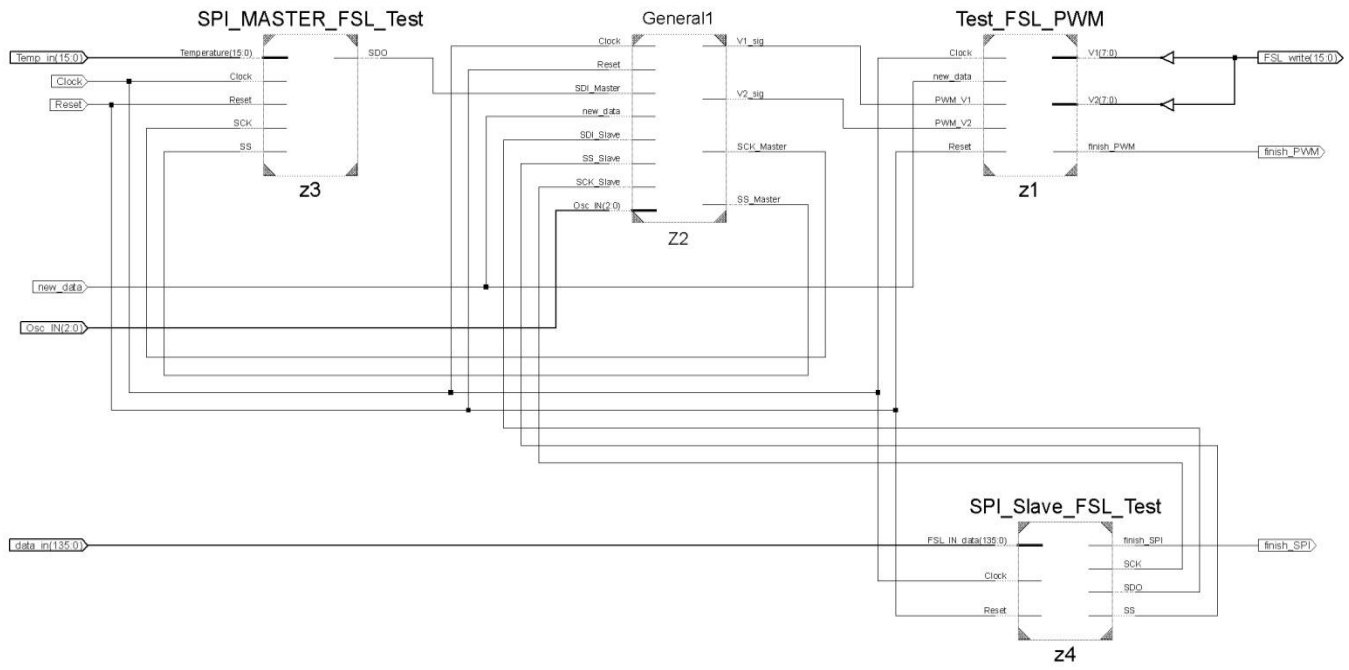

Figure 7: General Module connection

2.10 Test Benches

The test benches through the whole project are used in the same procedure as described here. The top module should be added and test vectors should be generated. There are two functions in the code that convert std_logic type to string and vice versa. In the first step, a line is read from the first file and appropriate numbers are sent to input ports of the top design module. In addition, the results are read from the second file and those are saved in register until the results of top module become ready. In the final stage, two results are compared and errors are printed on the screen. These loops are iterated until the end of the file.

2.11 Reader_Main.C

This file is the top module for FPGA test in the Xilinx Evaluation Board. The oscillation data are saved in this file as array and sent to the FSL module. After ports initialization, the data are sent through FSL and after a 256-cycle delay; the response is read and saved. At the end, results are written in the RS232 port for comparison with the expected data.

2.12 FSL.vhd

The FSL link is communicated between the C Module and VHDL codes through the FSL module. There are nine input ports and eight output ports that five of them are connected to the top VHDL module General. The definition of the ports is declared in the following table.

Ports	Definition
FSL_Clk	Synchronous clock
FSL_Rst	System reset, should always come from FSL bus
FSL_S_Clk	Slave asynchronous clock
FSL_S_Read	Read signal, requiring next available input to be read

FSL_S_Data	Input data
FSL_S_CONTROL	Control Bit, indicating the input data are control word
FSL_S_Exists	Data Exist Bit, indicating data exist in the input FSL bus
FSL_M_Clk	Master asynchronous clock
FSL_M_Write	Write signal, enabling writing to output FSL bus
FSL_M_Data	Output data
FSL_M_Control	Control Bit, indicating the output data are control word
FSL_M_Full	Full Bit, indicating output FSL bus is full
finish_SPI	ID and other SPI data is written
new_data	New frequency is detected
finish_PWM	PWM generation for one frequency is finished
bits_OSC	New detected frequency
FSL_write_to	Result data from top VHDL

Table 1. FSL Port Definition

Whenever the SPI task is done, the existence of input data from FSL is checked and if there is any, it is transferred to detected frequency and new_data port is set to high for one clock cycle. In the next stage, the Finish_PWM port is checked and whenever it is set to high, the result is read and written into FSL port.

2.13 FSL_Test.vhd

It connects General file to FSL in order to establish communication.

Chapter 3

THE TEST VECTOR GENERATORS

For testing VHDL codes, numerous number of test vectors are needed. For producing test vectors according to different phases of project, a number of different test vector generators such as State machine tester, PWM tester, TV_Gen, Temperature protocol tester, test1 and smtv have been developed.

3.1 State machine

State machine is the heart of TAG. For testing the State machine, three versions of the application have been developed. First for testing simple state machine without temperature protocol, second is for testing the temperature protocol which has been developed later, and final version for testing the whole system with the ability of defining and changing the values of v1 and v2 for each frequency and also the ability of defining the number of test vectors.

3.1.1 SMWOT

SMWOT (State machine Without Temperature) is a simple application for generating IDsAccording to the protocol. Each ID should have 48 bits. SMWOT creates 50 IDs in one file and extracts another file from the first file, exactly according to the protocol pattern. In this application v1 and v2 for each frequency have been predefined in the source of application and it does not consist of temperature protocol. Figure 8 shows the SMWOT application.

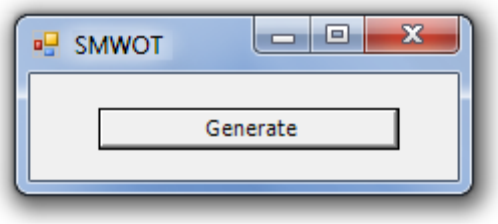

Figure 8. SMWOT Application

3.1.2 TPT

TPT stands for Temperature Protocol Tester, which is generates two files. One is same as the ID file and second file is extracted from first file according to the developed protocol.
The only difference is the number of bits which in this case is 16 bits, representing 16-bit temperature sensors output.

3.1.3 T.V.Gen.

Test Vector Generator application is a complete and fully functional application which has the ability of producing millions of test vectors for testing state machine. Unlike MSWOT and TPT, T.V.Gen. has ability of defining different values for F1, F2, F3, F4 and also v1 and v2 for each frequency. Full protocol of State machine has been tested by 20,000,000 test vectors which are generated by T.V.Gen. application.
Figure 9 and figure 10 show, T.V.Gen. application before and after generating the test vectors.

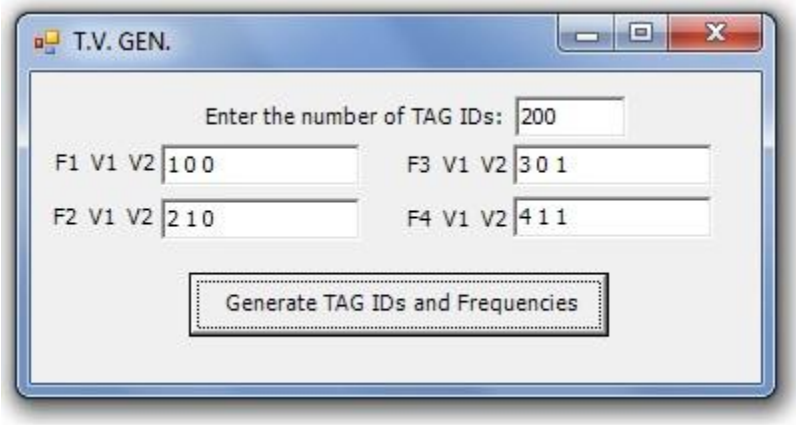

Figure 9. T.V.Gen. before starting to generate test vectors

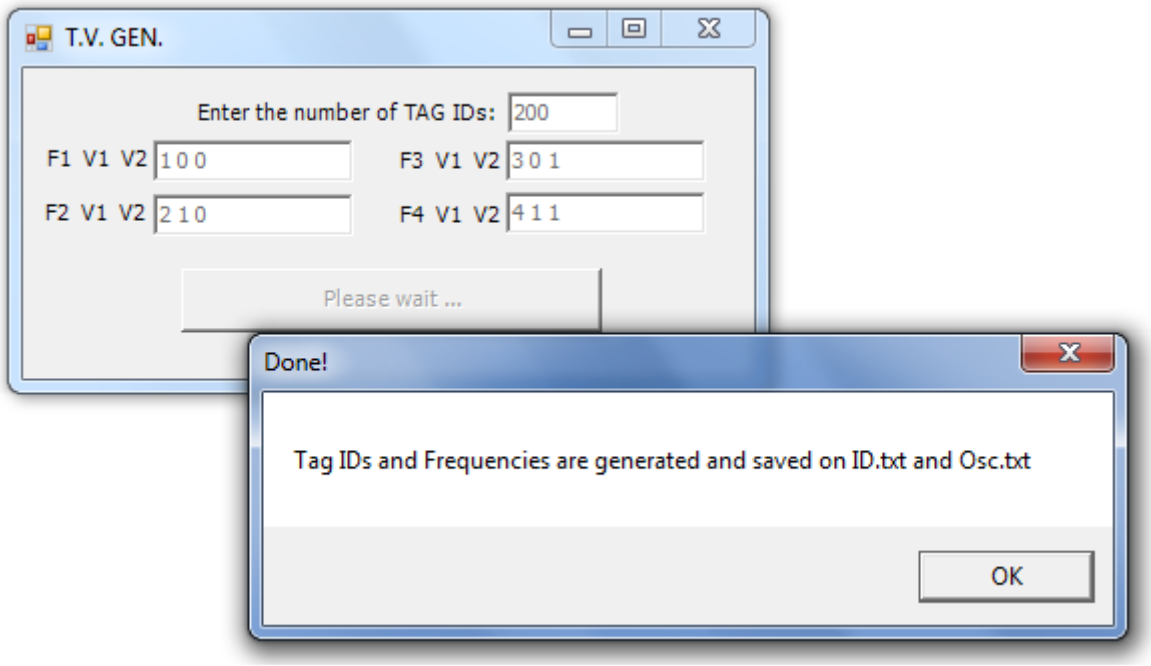

Figure 10. T.V.Gen. After Generating Test Vectors

In T.V.Gen. Application, Number of TAG IDs indicates the number of needed test vectors, and following fields are for entering the value for frequencies and appropriate V1 and V2 values for each frequency.

3.2 Pulse-width modulator tester (PWMT)

PWMT is an application for testing PWM part. In this application by entering number of test vectors, and pressing Generate PWM, 2 files will be generated. In first file, for each test vector line, two decimal numbers will be generated. In the other file 8-bit binary number for V1 and eight bit binary number for V2 will be generated. PWM module reads the first file as an input. Output of PWM module should be exactly the same as the second file's data. In this application user should only enter the number of test vectors. The Following figure shows the PWMT application window.

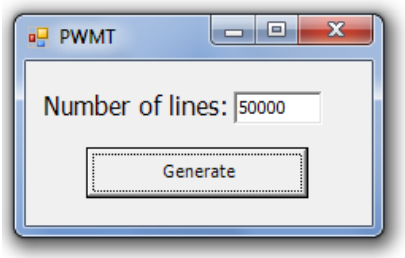

Figure 11. Window of PWMT application

3.3 Final Test Vector Generator

For testing the implemented TAG, 2 types of application have been developed. Manual Final Test Vector Generator for primary test of whole system and Auto. Final Test Vector Generator for testing the system with almost every possible condition. In following section complete description of each application is provided.

3.3.1 Manual Final Test Vector Generator (MFTVG)

MFTVG is a final test vector generator that covers State Machine, PWM, SPI Master and SPI Slave modules. This application is capable of generating millions of test vectors for testing the entire TAG operation. This application is using manually entered data for FxV1, FxV2, temperature and maximum sleep time to generate the test vectors. That means after generating IDs, the same pattern according to the entered data will be applied to generate the each line of output file. For example if F1V1 = 10001101 and F1V2 = 11111001, in each line of output file the same V1 and V2 will be replaced for F1. Figure 12 shows the window of MFTVG application with manually entered data.

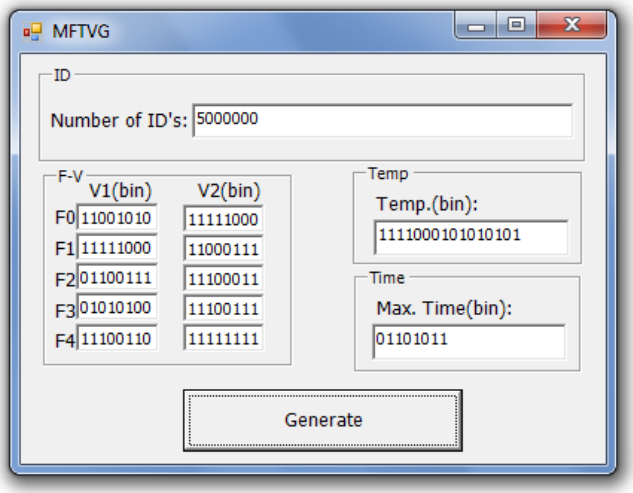

Figure 12. Window of MFTVG application after clicking Generate

3.3.2 Auto. Final Test Vector Generator (AFTVG)

This application's appearance looks like MFTVG application, but behind the scene everything is different. The only input of AFTVG is the number of needed test vectors. When the application starts to generate test vectors, instead of using pre-entered data, it generates a random value for each line of output data. This means unlike MFTVG, AFTVG uses different FxV1, FxV2, temperature and maximum sleep time for each generated test vector. By using AFTVG, operation of TAG in almost every possible condition has been tested. The illustration below shows the main window of AFTVG.

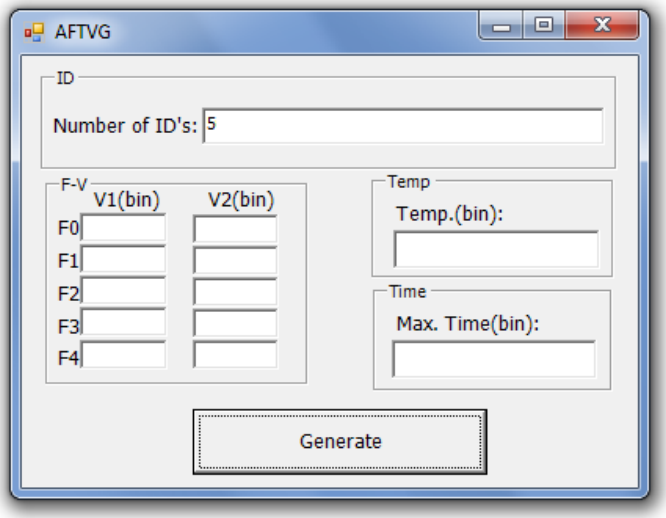

Figure 13. Window of AFTVG application after clicking Generate

Figure 14 shows the AFTVG application after operation completion.

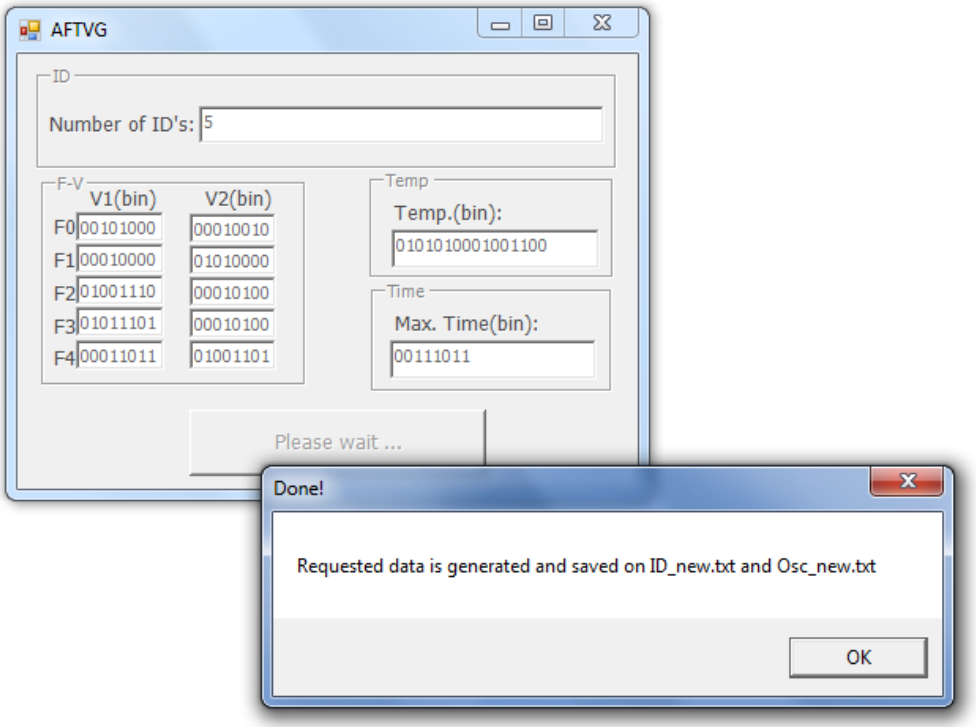

Figure 14. AFTVG application after finishing operation

Values in FxV1, FxV2, Temp. and Max. Time fields are the last random generated values for these variables.

Chapter 4

THE VHDL CODE IMPLEMENTATION AND VERIFICATION

The Tag Codes involve PWM_generator, State_machine, SPI_SLAVE, SPI_master_2. The connection between codes is demonstrated in the following figures. In this chapter, the testing method and FPGA implementation is discussed.

The code is implemented in Modelsim and each part is tested and verified through different test vectors. Then the code is implemented in FPGA.

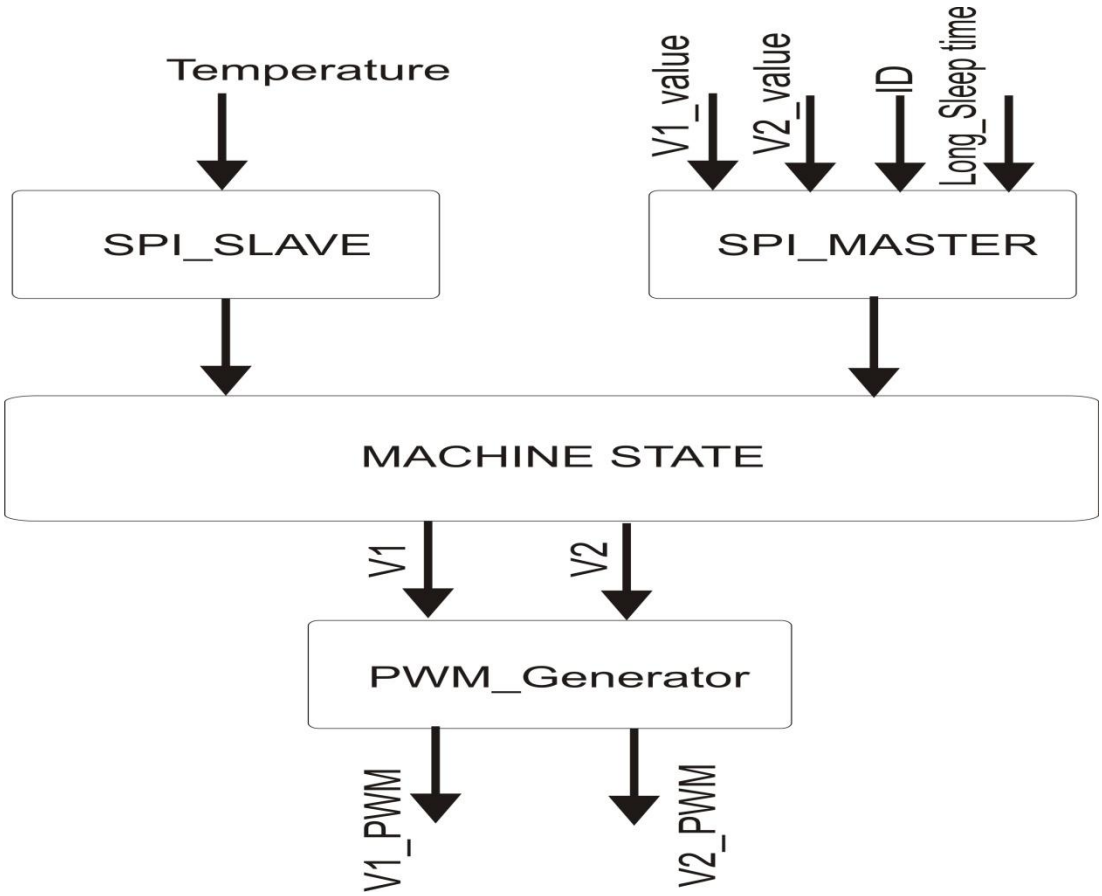

Figure 15. Tag's Code connection

4.1 MODELSIM

For each part of the code, a Test_Bench is written and according to requirements, test vectors are prepared. With the aid of test vectors the inputs are fed and expected outputs are checked. In the case of inconsistency, the problems are found and corrected.

The state machine is tested with twenty million test vectors contained in two test vector files. The method of testing is simple, from the first file an ID and a detected frequency is read and with previous associated V1 and V2 values for Frequency, the output is generated and compared with the second file. The Testing method is accurate because all individual states of state_machine are tested and observed automatically.

The PWM_generator part is responsible to generate PWM based on the defined resolution. Two input numbers are propagated and the outputs are generated and compared with the test vector file which contains the state of the output in the each cycle.

Also, in the SPI components (SPI_MASTER, SPI_SLAVE) the same procedure is repeated and a file which contains temperature or ID and values of output frequencies are propagated through inputs and in each cycle expected state is read and compared with generated state which contains expected state in that cycle is compared.

In order to test the entire files‘consistency, a new test method is utilized. To be able to read the PWM outputs, a new module Test_FSL_PWM is introduced. This module receives PWM and in the Defined Resolution builds a number which represents the number of cycles which PWM signal is high. In other words, it acts in the opposite order of the PWM_generator module. For the other modules, SPI_SLAVE and SPI_MASTER, the same procedure is used. For Example, SPI_SLAVE_FSL_test is used to act in the opposite order of the SPI_SLAVE procedure. It should be noted that SPI_Slave and SPI_MASTER is written to cooperate with external Device U2C-12 USB and sensor LM74. Clearly, The SPI_SLAVE_TEST and SPI_MASTER_TEST have simulated U2C-12 USB and LM74 respectively. The Figure in the next page shows portion of MODELSIM window in the project.

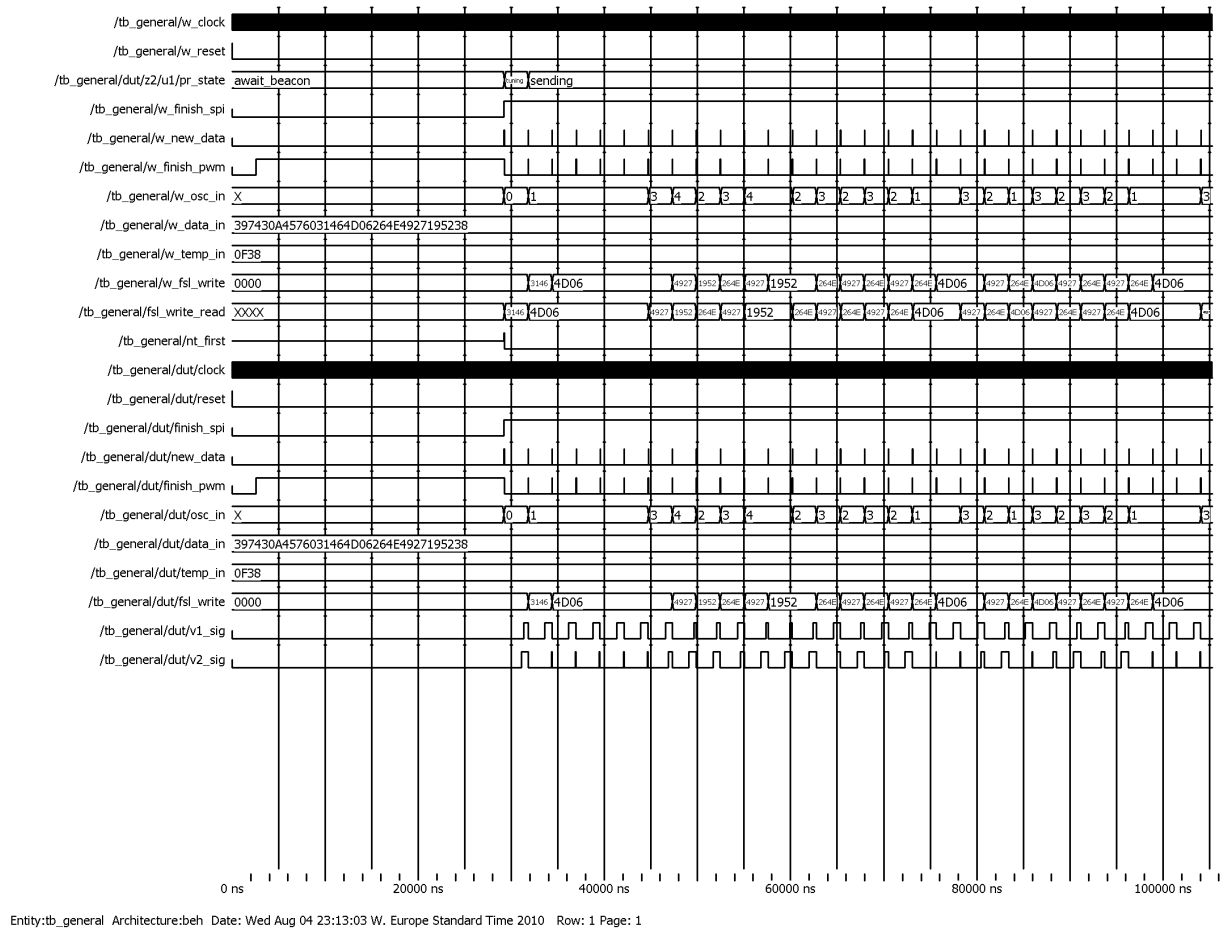

Figure 16. ModelSim Waveform Review

4.2 Hardware interface

In order to send data such as ID, V1, V2 values and … to TAG, the USB-I2C/SPI/GPIO Interface Adapter is used. The adapter communicated with tag through SPI interface and is connected to computer with a USB interface. For this purpose the Diolan USB-I2C/SPI is used. It should be noted that the device needs at least 2v for high level voltage [3], which means a voltage convertor is needed between tag and adapter because tag works in 1.2 V .The adapter is shown in the next figure.

Figure 17. Diolan Adapter [3]

It is possible to modify the adapter to SPI mode 0 and after setting the basic properties such as clock for SPI, the adapter sends the entire 136 bits of its data to tag. For more information refer to DataSheet in the appendix.

To receive temperature, the LM74 is used. The Lm74 sends data in 2 bytes with SPI protocol in Slave mode. Therefore, the tag is works as master and repeatedly reads data and sends it to Reader. It should be noted that the data is sent to reader in the same order as read from LM74, in other words reader should convert it to human readable numbers. To LM74 needs 2.65 V to operate, therefore DC-DC voltage convertor is needed. Refer to datasheet of LM74 for more details which is provided in the appendix.

Because both of above-mentioned devices were not provided at the time of project, The SPI_MASTER_FSL_TEST module as LM74 and SPI_SLAVE_FSL_TEST module as adapter are used.

To test and verify the VHDL Code, Spartan-3E starter KIT is used. The main reason to choose this board is because it is supported in ISE 12 and the computer which was used in this project had Windows 7 installed which only supports ISE 11 and ISE 12. The board has a built-in clock of 50MHz and it support MicroBlaze microprocessor [4].

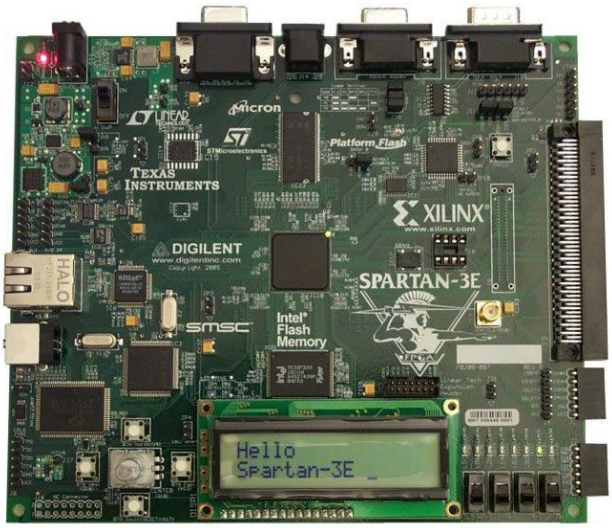

Figure 18. Spartan 3-E [4]

4.3 FPGA implementation

The top vhdl module is connected to FSL links to communicate with the MicroBlaze processor. The C program in MicroBlaze is simple and it sends stored data to FSL and receives data and sends data through serial port to the terminal.

There was two different ways in storing data in the FPGA and send them to serial port. The first method is to store input and output data in the FPGA and send it one by one. In this case more than 1000 words must be stored but it has the advantage to use FSL example module of the MicroBlaze. In first try, the above approach is used but the FPGA has only 10,000 cells and it was not feasible to store such high volume of data, therefore the second approach is used [4]. In the second method, data is send and respond is received and sent to FSL. This approach is more difficult because it needs synchronization between different parts which is not possible with pre-defined FSL module. The new FSL module is utilized and the other modules must be synchronized. Synchronization is done through two signals. The first signal which is outputted from SPI modules indicates when the SPI writing is finished and second signal which is an output of PWM modules and indicates when the output is ready. It should be considered that the first approach has the advantage of sending new ID to the Tag but in the second approach the ID should be stored in the VHDL files.

Chapter 5

SYNTHESIS, VERIFICATION AND POWER ANALYSIS

In the following sections VHDL code synthesis and verification of the synthesized netlist will be discussed.

5.1 Synthesis

After running RTL compiler and adding 130-nm technology library, vhdl files should be added and elaborated. These files are My_data_type.vhd, Machine_state.vhd, PWM_Generator.vhd, SPI_MASTER_2.vhd SPI_SLAVE.vhd, my_components.vhd and General1.vhd.
For implementing hardware descriptions on real hardware, synthesis is needed. To get an idea about the timing constraints, synthesizing without any timing constraint is essential. Synthesizing with no timing constraint shows that the best synthesis effort and timing constraint for this project are medium effort and 1200ps respectively. After this step the netlist has been extracted. The Following table shows the timing slack, area and power of design after synthesis.

Clock constraint = 1200ps | Synthesis effort = Medium		
Timing Slack	**Area**	**Power**
0 ps	25598.268 μm^2	10132529.064 nW

Table 2. Primary data of design

5.2 Netlist verification

After synthesizing the vhdl code, the netlist that actually is a verilog file, needs to be verified against the original VHDL code. This can be done by forcing RTL compiler to produce netlist and verifying it by the same testbench that is used for verifying the original vhdl code. Passing this step is essential to ensure that the design functions as desired.
To achieve this, we have to read VHDL libraries that describe the standard cells, the verilog file which is extracted in the synthesis phase, and TEST_FSL_PWM.vhd,SPI_Slave_FSL_Test.vhd, SPI_Master_FSL_Test.vhd, my_Component_2.vhd, General.vhd, General_tb.vhd.
In this phase the design has been verified by using almost 1,000,000 test vectors.

5.3 Power analysis

To find the exact power consumption of the design, simulation directives have to be altered to enable the creation of the VCD (Value Change Dump) file. In later stages, the RTL compiler will use this VCD file to compute the switching statics by applying the provided test vectors.
The VCD file has been used to produce the new power report. The Following table shows the accurate power of each block.

Extracted power from VCD file			
Block Name	***Leakage Power(nW)***	***Dynamic Power(nW)***	***Total Power(nW)***
State Machine	298130.099	209926.390	508056.489
PWM Generator	207078.997	176921.524	384000.521
SPI Master	109890.165	283633.387	393523.551
SPI Slave	548233.893	1281225.474	1829459.368
Total	1163731.857	1951754.918	3115486.775

Table 3. Accurate power of each block, using VCD file

Chapter 6

PLACE & ROUTE

As indicated by the name, Place and Route consists of placement and routing. The first step is concerned about deciding where to put the different blocks and the second step tries to connect these different blocks together [5].

In the following sections floorplaning and routing will be described.

6.1 Floorplaning

In this step different blocks of the design have been placed in an appropriate order. Figure 19 shows the order of block placement.

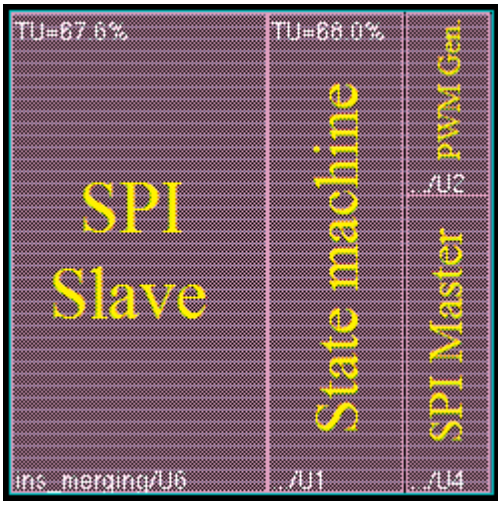

Figure 19. Blocks order

6.2 Routing

In the routing step power rings have been placed around the core. For getting a more accurate timing report, in Clock Tree Synthesis (CTS) phase, clock tree has been created. To eliminate the timing violations in the design, different orders of block placement have been investigated. The best order, was shown in figure 19.
The gaps between blocks, have been filled up by fillers, and at the end, the entire design has been verified.

6.3 Pin description

This design has 16 I/O pins. 2 pins for Vdd and Gnd, and 14 pins delivering the functionality of the Tag.
A brief description of each pin is provided in table 3.

No.	Pin	Direction	Description
01	Reset	Input	Reset pin (enable high)
02	Clock	Input	Clock
03	New_Data	Input	Incoming new data indicator
04	Osc0	Input	Bit0 of detected frequency
05	SIG_V2	Output	PWM2
06	SIG_V1	Output	PWM1
07	Master_SCK	Input	Clock for SPI(master)
08	Gnd	Input	Ground
09	Master_SS	Input	Enable SPI(master)
10	Master_SDI	Input	Data for SPI(master)
11	Osc2	Input	Bit1 of detected frequency
12	Osc1	Input	Bit2 of detected frequency
13	Slave_SCK	Input	Clock for SPI(slave)
14	Slave_SDI	Input	Data for SPI(slave)
15	Slave_SS	Input	Enable SPI(slave)
16	Vdd	Input	Power

Table 4. Pin description of design

Figure 20 shows the final design and pin names.

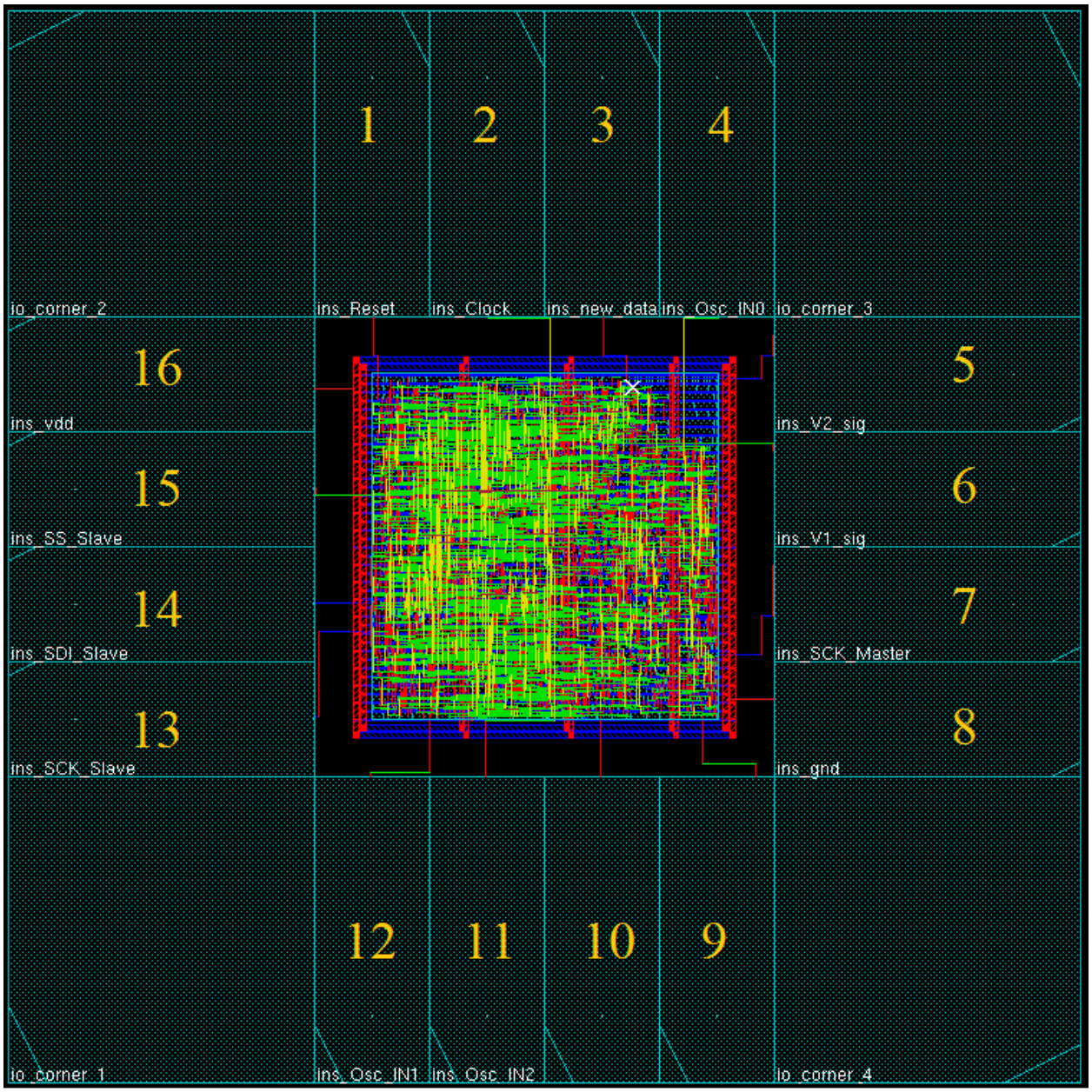

Figure 20. Final designed chip

CONCLUSION

As a conclusion, the frequency of this design is almost 833MHz, power consumption is 3.1 mW and occupied area is almost 25 mm^2. As it has shown in figure 19 and Table 3, the major area of the core is occupied by the SPI slave module, which leads to more power consumption. Using I^2C protocol was another alternative, but because of the following two reasons using SPI protocol was preferable.

1. I^2C has fewer lines, which means less pins, but it needs pull-up resistors in order to maintain the high level, which translates to higher power consumption.

2. Due to nature of the full duplex communication, the speed of SPI protocol is higher than I^2C that leads to reach higher frequency [8].

As the SPI slave module is in use just during the feeding data into the tag, the power consumption of this block can be neglected. However as the low power consumption and higher speed was our priority, the SPI protocol was chosen for this design.

The following data is the overall specifications of the design.

Frequency: 833 MHz
Power (w/o SPI Slave): 1.28 mW
Power (Total): 3.11 mW
Area: 25.1mm^2
Communication Protocol: SPI Mode 0
Temp. Sensor: LM74
Pins: 16

BIBLIOGRAPHY

[1]. EGON protocol reference, Lepton Radio AB

[2]. Serial Peripheral Interface.
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
Accessed on 2010-08-01

[3]. USB - I2C/SPI adapter - U2C-12
http://www.diolan.com/i2c/u2c12.html?uie&gclid=CK_p5qLz458CFVOPzAodMXUMHQ
Accessed on 2010-08-03

[4]. Spartan-3E FPGA Starter Kit Board User Guide, Xilinx, June 2008

[5]. Place and Route
http://en.wikipedia.org/wiki/Place_and_route
Accessed on 2010-08-11

[6]. LM74 datasheet
http://www.national.com/ds/LM/LM74.pdf
Accessed on 2010-08-03

[7]. Radio Frequency Identification
http://en.wikipedia.org/wiki/Radio-frequency_identification
Accessed on 2010-08-27

[8]. I2C or SPI serial communication.
http://dev.emcelettronica.com/i2c-or-spi-serial-communication-which-one-to-go
Accessed on 2010-08-27

Appendix A

Source code of VHDL files

My_data_type.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Package My_data_type IS
Type Frequency is (F0,F1,F2,F3,F4);
END Package;
```

Machine_state.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USe work.My_data_type.all;
-------------------------------------------------------------
ENTITY Machine_state IS
Generic (MAX_BIT : Integer :=48;MAX_Temp:integer:=16;PWM_MAX:integer:=256) ;
----maximum  bIt will alwasy 48 but max_sleep should be change and max_Temp
----maxmum Tempertuare bits is fixed will be 16
port(Clock,Reset: IN std_logic;
new_data: in std_logic;
ID: IN std_logic_vector(MAX_BIT-1 downto 0);
MAX_sleep : IN std_logic_vector(7 downto 0) ;
Temp : IN std_logic_vector(MAX_Temp-1 downto 0);
Osc_detect: IN frequency;
V1_VALUE_F0:IN  std_logic_vector(7 downto 0);
V2_VALUE_F0:IN  std_logic_vector(7 downto 0);
V1_VALUE_F1:IN  std_logic_vector(7 downto 0);
V2_VALUE_F1:IN  std_logic_vector(7 downto 0);
V1_VALUE_F2:IN  std_logic_vector(7 downto 0);
V2_VALUE_F2:IN  std_logic_vector(7 downto 0);
V1_VALUE_F3:IN  std_logic_vector(7 downto 0);
V2_VALUE_F3:IN  std_logic_vector(7 downto 0);
V1_VALUE_F4:IN  std_logic_vector(7 downto 0);
V2_VALUE_F4:IN  std_logic_vector(7 downto 0);
V1: OUT std_logic_vector(7 downto 0);
V2: OUT std_logic_vector(7 downto 0));
END Machine_state;
-------------------------------------------------------------
ARCHITECTURE behave OF Machine_state IS
TYPE state IS (sending,tuning,await_beacon,Send_temp,long_sleep);
SIGNAL pr_state:state;
--SIGNAL Osc_detect_tmp:Frequency;
function std_logic_to_integer(inp: std_logic_vector(7 downto 0)) return integer  is
variable tmp: integer range  0 to 255;
```

```
begin
if (inp(0) = '1') then
tmp := 1  ;
else
tmp :=0   ;
end if;
if (inp(7) = '1') then
tmp := tmp + 128  ;
end if;
if (inp(6) = '1') then
tmp := tmp + 64  ;
end if;
if (inp(5) = '1') then
tmp := tmp + 32  ;
end if;
if (inp(4) = '1') then
tmp := tmp + 16  ;
end if;
if (inp(3) = '1') then
tmp := tmp + 8  ;
end if;
if (inp(2) = '1') then
tmp := tmp + 4  ;
end if;
if (inp(1) = '1') then
tmp := tmp + 2  ;
end if;
return tmp;
end function std_logic_to_integer;
BEGIN
PROCESS(Reset,Clock) ---
variable cnt_ID : integer RANGE  0 to MAX_BIT;
variable cnt_Temp : integer RANGE  0 to MAX_Temp-1;
variable cnt_sleep : integer RANGE  0 to 255;
variable MAX_sleep_tmp:integer range 0 to 255;
variable Temp_tmp:std_logic_vector(1 downto 0);
variable ID_temp:std_logic_vector(1 downto 0);
variable Temp_start:std_logic;
BEGIN
IF(Clock'event and Clock='1')  then
if(Reset='1')then
pr_state<=await_beacon;---initialize ports
V1<=x"FF";
V2<=x"FF";
cnt_ID:=0;
cnt_Temp:=0;
cnt_sleep:=0;
MAX_sleep_tmp:=255;
elsif (new_data='1' or pr_State=long_sleep) then
--if(new_data='1' or pr_State=long_sleep)then
CASE pr_State IS
```

```
WHEN Send_temp =>
if( Osc_detect=F2 and Temp_start='0' )then---problem here--Check if reader wants TEMP
cnt_Temp:=0;
V1 <= V1_VALUE_F2;
V2 <= V2_VALUE_F2;
Temp_start:='1';
elsif(  Temp_start='1'  )then  --- if it need start to send temp
if(Osc_detect/=F2 and cnt_Temp>1)then ----if reader doesn't give
cnt_Temp:=cnt_Temp-1;        ------ACK send the bit again
--- here v1 and v2 is previous one,maybe error occur
end if;
if(cnt_temp < MAX_Temp )then ----Start to send each bit(FROM MSB to LSB)
--(Consider 11010100. For first bit(0(cnt_temp-1=MSB))=1 and Bit(1)=1 is )
if(cnt_temp = MAX_Temp-1)then  ---if all bits send go to sleep
V1 <= V1_VALUE_F2;--X"01";  ---FF-01
V2 <= V2_VALUE_F2;--X"00";  ---FF-00
Temp_start:='0';
pr_state<= long_sleep;
else
Temp_tmp(0):=Temp(cnt_Temp);
Temp_tmp(1):=Temp(cnt_Temp+1);
if (Temp_tmp="00")then
V1 <= V1_VALUE_F1;
V2 <= V2_VALUE_F1;
elsif(Temp_tmp="10")then
V1 <= V1_VALUE_F2;
V2 <= V2_VALUE_F2;
elsif(Temp_tmp="01")then
V1 <= V1_VALUE_F3;
V2 <= V2_VALUE_F3;
elsif(Temp_tmp="11")then
V1 <= V1_VALUE_F4;
V2 <= V2_VALUE_F4;
end if;
cnt_Temp:=cnt_Temp+1; ---- counter for TEMP BITS
end if;
else   ---- if there is other frequency than F2 , no temp back to sleep
cnt_Temp:=0;
V1 <= X"FF";
V2 <= X"FF";
pr_state<= long_sleep;
end if;
WHEN sending =>   ---sending section, Response to ID which is send by reader
--- 4 first if define the V1 and V2 on output in response to DETECTION
ID_temp:=ID(cnt_ID+1 downto cnt_ID);
if( ID_temp(0)='0' and ID_temp(1)='0' and Osc_detect=F1 )  then
V1 <= V1_VALUE_F1;
V2 <= V2_VALUE_F1;
cnt_ID:=cnt_ID+1;
elsif( ID_temp(0)='1' and ID_temp(1)='0' and Osc_detect=F2 )  then
V1 <= V1_VALUE_F2;
```

```
V2 <= V2_VALUE_F2;
cnt_ID:=cnt_ID+1;
elsif( ID_temp(0)='0' and ID_temp(1)='1' and Osc_detect=F3 )  then
V1 <= V1_VALUE_F3;
V2 <= V2_VALUE_F3;

cnt_ID:=cnt_ID+1;
elsif( ID_temp(0)='1' and ID_temp(1)='1' and Osc_detect=F4 )  then
V1 <= V1_VALUE_F4;
V2 <= V2_VALUE_F4;
cnt_ID:=cnt_ID+1;
elsif( Osc_detect=F0 )  then   ----IF Suddenly F0 receive
pr_state<= sending;    --- remain in sending
else
pr_state<=await_beacon;  ----any wrong Frequency according to
cnt_ID:=0;  --- ID go to Await beacon wait for F0
end if;
if(cnt_ID = MAX_BIT-1)then  --- IF all ID bits check go to send Temp
cnt_ID:=0;
cnT_temp:=0;
Temp_start:='0';
pr_state<=Send_temp;
end if;
WHEN tuning =>  ---- in this state reader choose ID and
-----if it's our tag ID go to Sending state
if    ( (ID(1)='0' and ID(0)='0' and Osc_detect=F1)    ) then
cnt_ID:=0;
pr_state<= sending;
V1 <= V1_VALUE_F1;
V2 <= V2_VALUE_F1;
elsif (ID(1)='0' and ID(0)='1' and Osc_detect=F2) then
cnt_ID:=0;
pr_state<= sending;
V1 <= V1_VALUE_F2;
V2 <= V2_VALUE_F2;
elsif (ID(1)='1' and ID(0)='0'and Osc_detect=F3) then
cnt_ID:=0;
pr_state<= sending;
V1 <= V1_VALUE_F3;
V2 <= V2_VALUE_F3;
elsif  (ID(1)='1' and ID(0)='1'and Osc_detect=F4)  then
cnt_ID:=0;
pr_state<= sending;
V1 <= V1_VALUE_F4;
V2 <= V2_VALUE_F4;
else  --- if it dosen't choose ours go to await beacon
pr_state<=await_beacon;
End if;
WHEN await_beacon =>  ----Check if receive beacon
if( Osc_detect = F0 ) then
pr_state<=tuning;
```

```
V1 <= V1_VALUE_F0;
V2 <= V2_VALUE_F0;
else ----IF there is no beacon wait
pr_state<=await_beacon;
V1 <= X"FF";
V2 <= X"FF";
end if;
WHEN long_sleep =>   -----send all the data now go for long sleep

V1 <= X"FF";   ----after long sleep go to await beacon
V2 <= X"FF";
if (cnt_sleep=((MAX_sleep_tmp)))then
pr_state<=await_beacon;
cnt_sleep:=0;
elsif(Cnt_sleep=0)then
cnt_sleep:=1;
MAX_sleep_tmp:=std_logic_to_integer(MAX_sleep) ;
if(MAX_sleep_tmp=0)then
MAX_sleep_tmp:=1;
end if;
else
pr_state<=long_sleep;
cnt_sleep:=cnt_sleep+1;
end if;
end case;
end if;
end if;
END process;
-------------------------------------------
ENd behave;
```

SPI_MASTER_2.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------------
ENTITY SPI_MASTER_2 IS
Generic (MAX_Time : Integer :=10; MAX_Temp_BITS: Integer :=16 ) ;
port(Clock,Reset,SDI: IN std_logic;
--MAX_Temp_BITS : IN integer range 0 to 255;
SCK: OUT std_logic;
SS : OUT std_logic;
TEMP: OUT std_logic_vector(MAX_Temp_BITS-1 downto 0));
END SPI_MASTER_2;
-------------------------------------------------------------
ARCHITECTURE behave OF SPI_MASTER_2 IS
Signal start:std_logic;
BEGIN
Process(Reset,Clock)
```

```
Variable Cnt_Bits:integer range 0 to MAX_Temp_BITS; ---- maximum bits to recive
variable Cnt_CLk:integer  range 0 to (Max_Time*4+1);   --- counter to generate SCK
variable Reg_Temp:std_logic_vector(MAX_Temp_BITS-1 downto 0);----temo register to save data
variable Tmp_SCK:std_logic;---- Save previous generated clock
variable Counter_Byte:integer range 0 to 10; --- counter if it reach a bye or not
variable extra_delay:std_logic;  ---- rgister to define extra delay between bytes
begin
if(Reset='1')then

---initialize all reg
Cnt_Clk:=0;
SCK<='0';
Tmp_SCK:='0';
SS<='1';
Cnt_Bits:=0;
extra_delay:='0';
start<='0';
Counter_Byte:=0;
Reg_Temp:=(others=>'0');
TEMP<=(others=>'0');
ELSIF(Clock'event and Clock='1'  )then
Cnt_Clk:=Cnt_Clk+1;
if( Start='0' and Cnt_clk=Max_Time*2)then  --- wait for 2 at least time after each receive for settle time
Start<='1'; ---- start to recieve
ss<='0';---put the start signal for sensor
Cnt_Clk:=0;--- start to get times
elsif(start='1' and Cnt_clk=Max_Time and Counter_Byte<8 and Cnt_Bits<MAX_Temp_BITS)then
SS<='0';
Cnt_Clk:=0;--set for another bit
if(Tmp_SCK='1')then
Tmp_SCK:='0'; ---Falling __data change on falling edge
else              ---RISING  __data read on rising edge
Tmp_SCK:='1';
Reg_Temp(MAX_Temp_BITS-Cnt_Bits-1):=SDI; --data read from sensor
Cnt_Bits:=Cnt_Bits+1;
Counter_Byte:=Counter_Byte+1;---count if we get one byte for extra delay
end if;
extra_delay:='0';  ---no delay , just after each recieve because of
SCK<=Tmp_SCK;  ----set CLOCK for spi
elsif(start='1' and Cnt_clk=Max_Time and Counter_Byte=8 and extra_delay='0')then
---- a byte recieve so add extra max_time delay
extra_delay:='1';
Tmp_SCK:='0';
SCK<='0';
Cnt_Clk:=0;
elsif(start='1' and Cnt_clk=Max_Time and Cnt_Bits=MAX_Temp_BITS)then
----all bits receive make reg ready for next one
Cnt_Bits:=0;
Counter_Byte:=0;
SS<='1';
start<='0';
```

```
Cnt_Clk:=0;
TEMP<=Reg_Temp;
elsif(start='1' and Cnt_clk=Max_Time and Counter_Byte=8 and extra_delay='1')then
extra_delay:='0';  ----extra delay employed , back to recive next one
Cnt_Clk:=0;
Counter_Byte:=0;
SCK<='0';
end if;
end if;
end process;
end behave;
```

SPI_SLAVE.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------------------------
ENTITY SPI_SLAVE IS
Generic (MAX_BIT :integer := 136 ; ID_bit:integer:=48 ) ;
port(Clock,Reset,SDI,SCK,SS: IN std_logic;
--SDO: OUT std_logic;
--MAX_ID_BITS : OUT std_logic_vector(7 downto 0);--0 to 127
ID: OUT std_logic_vector(ID_bit-1 downto 0);
V1_VALUE_F0:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F0:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F1:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F1:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F2:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F2:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F3:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F3:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F4:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F4:OUT  std_logic_vector(7 downto 0);
MAX_SLEEP_TIME: OUT std_logic_vector(7 downto 0) );
--MAX_TEMP: out std_logic_vector(7 downto 0)
END SPI_SLAVE;
---------------------------------------------------------------
ARCHITECTURE behave OF SPI_SLAVE IS
signal toggle_SCK,Data_ready:std_logic;
BEGIN
Process(Reset,Clock)
variable TMP:std_logic_vector(MAX_BIT-1 downto 0);
variable cnt_bits:integer range 0 to MAX_BIT;
BEGIN
if(Reset='1')then
ID<=X"80CE64EEAD54" ;
V1_VALUE_F0<=X"11";
V2_VALUE_F0<=X"11";
V1_VALUE_F1<=X"00";
V2_VALUE_F1<=X"00";
V1_VALUE_F2<=X"01";
```

```
V2_VALUE_F2<=X"00";
V1_VALUE_F3<=X"00";
V2_VALUE_F3<=X"01";
V1_VALUE_F4<=X"01";
V2_VALUE_F4<=X"01";
MAX_SLEEP_TIME<=X"20";
-- MAX_TEMP<=X"08";
cnt_bits:=0;--- number of bit recieve store
TMP(MAX_BIT-1 downto 0):=(others=>'0'); ----temp register to store data
Data_ready<='0';  ---- if all data recive issue '1'
toggle_SCK<='1';  ---- initialization for register which store data
ELSIF(Clock'event and Clock='1' )then
if(data_ready='1')then ---- if all data get read issue it to output
data_ready<='0';
ID<=TMP( MAX_BIT-1 downto MAX_BIT-ID_bit) ;
V1_VALUE_F0<=TMP( MAX_BIT-ID_bit-1 downto MAX_BIT-ID_bit-8);
V2_VALUE_F0<=TMP( MAX_BIT-ID_bit-9 downto MAX_BIT-ID_bit-16 );
V1_VALUE_F1<=TMP( MAX_BIT-ID_bit-17 downto MAX_BIT-ID_bit-24 );
V2_VALUE_F1<=TMP( MAX_BIT-ID_bit-25  downto MAX_BIT-ID_bit-32 );
V1_VALUE_F2<=TMP( MAX_BIT-ID_bit-33 downto MAX_BIT-ID_bit-40);
V2_VALUE_F2<=TMP( MAX_BIT-ID_bit-41 downto  MAX_BIT-ID_bit-48);
V1_VALUE_F3<=TMP( MAX_BIT-ID_bit-49 downto MAX_BIT-ID_bit-56 );
V2_VALUE_F3<=TMP( MAX_BIT-ID_bit-57 downto MAX_BIT-ID_bit-64 );
V1_VALUE_F4<=TMP( MAX_BIT-ID_bit-65 downto MAX_BIT-ID_bit-72 );
V2_VALUE_F4<=TMP( MAX_BIT-ID_bit-73 downto MAX_BIT-ID_bit-80 );
MAX_SLEEP_TIME<=TMP(MAX_BIT-ID_bit-81 downto 0);
end if;
if(ss='0')then  ---start id SS='0'
if(SCK='1' and toggle_SCK='0' and data_ready='0')then  ---read on Rising edge
TMP(MAX_BIT-cnt_bits-1):= SDI; --store data on sdi
cnt_bits:=cnt_bits+1;
if(cnt_bits=MAX_BIT)then --- if all 152 receive issue data ready
data_ready<='1';
cnt_bits:=0;
end if;
end if;
else  ---- if SS='1' initialize regsiters
cnt_bits:=0;
TMP(MAX_BIT-1 downto 0):=(others=>'0');
Data_ready<='0';
toggle_SCK<='1';
end if;
---- every time store clock data to detect rising and falling edge
if(SCK='1')then
toggle_SCK<='1';
else
toggle_SCK<='0';
end if;
end if;
END process;
-----------------------------------------
END behave;
```

PWM_Generator.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------------
ENTITY PWM_Generator IS
Generic (MAX_Time : Integer :=256) ;
port(Clock,Reset: IN std_logic;
new_data : in std_logic;
V1: IN std_logic_vector(7 downto 0) ;
V2 : IN std_logic_vector(7 downto 0) ;
PWM_V1 : OUT std_logic;
PWM_V2 : OUT std_logic);
END PWM_Generator;
-------------------------------------------------------------
ARCHITECTURE behave OF PWM_Generator IS
---- afunction to convert std_logic to integer so simple
function std_logic_to_integer(inp: std_logic_vector(7 downto 0)) return integer  is
variable tmp: integer range  0 to 255;
begin
if (inp(0) = '1') then
tmp := 1  ;
else
tmp :=0   ;
end if;
if (inp(7) = '1') then
tmp := tmp + 128  ;
end if;
if (inp(6) = '1') then
tmp := tmp + 64  ;
end if;
if (inp(5) = '1') then
tmp := tmp + 32  ;
end if;
if (inp(4) = '1') then
tmp := tmp + 16  ;
end if;
if (inp(3) = '1') then
tmp := tmp + 8  ;
end if;
if (inp(2) = '1') then
tmp := tmp + 4  ;
end if;
if (inp(1) = '1') then
tmp := tmp + 2  ;
end if;
return tmp;
end function std_logic_to_integer;
BEGIN
Process(Reset,Clock)
variable Cnt_PWM:integer range 0 to MAX_Time-1;
variable V1_tmp,V2_tmp:std_logic_vector(7 downto 0) ;
```

```
variable TMP:integer range 0 to 255;
begin
if(Reset='1')then
PWM_V1<='0';
PWM_V2<='0';
Cnt_PWM:=0;
V1_tmp:="00000000";
V2_tmp:="00000000";
ELSIF(Clock'event and Clock='1'  )then
if(new_data='1' )then ---and first_new_data='0'
PWM_V1<='0';
PWM_V2<='0';
Cnt_PWM:=0;
end if;
if(Cnt_PWM=MAX_Time-1)then
PWM_V1<='0';
PWM_V2<='0';
--finish_PWM<='1';
Cnt_PWM:=0;
end if;
IF((V1_tmp /= V1) or (V2_tmp /= V2) ) then --- see if the value is change start timer
Cnt_PWM:=0;
PWM_V1<='0';
PWM_V2<='0';
--finish_PWM<='0';
end if;
V1_tmp:=V1;
V2_tmp:=V2;
TMP:= std_logic_to_integer(V1_tmp);  ---convert to integer
if(TMP /=0 and  TMP /= MAX_Time-1)then
if(Cnt_PWM= (MAX_Time - TMP-1))then   --- for max-V1 or max-V2 time 0
PWM_V1<='1';
end if;
else
PWM_V1<='0';
end if;
TMP:= std_logic_to_integer(V2_tmp);
if(TMP /=0 and TMP /= MAX_Time-1)then
if(Cnt_PWM=(MAX_Time -TMP-1) )then
PWM_V2<='1';
end if;
else
PWM_V2<='0';
--finish_PWM<='0';
end if;
Cnt_PWM:=Cnt_PWM+1; --- add each time for both PWM OUTPUT SIGNAL
end if;
end process;
end behave;
```

my_components.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USe work.My_data_type.all;
---------------------------------
Package My_components Is
-------------------------
Component Machine_state IS
Generic (MAX_BIT : Integer :=48;MAX_Temp:integer:=16;PWM_MAX:integer:=256) ;
-----maximum  bIt will alwasy 48 but max_sleep should be change and max_Temp
-----maxmum Tempertuare bits is fixed will be 16
port(Clock,Reset: IN std_logic;


new_data  : in std_logic;
ID: IN std_logic_vector(MAX_BIT-1 downto 0);
MAX_sleep : IN std_logic_vector(7 downto 0) ;
Temp : IN std_logic_vector(MAX_Temp-1 downto 0);
Osc_detect: IN frequency;
V1_VALUE_F0:IN  std_logic_vector(7 downto 0);
V2_VALUE_F0:IN  std_logic_vector(7 downto 0);
V1_VALUE_F1:IN  std_logic_vector(7 downto 0);
V2_VALUE_F1:IN  std_logic_vector(7 downto 0);
V1_VALUE_F2:IN  std_logic_vector(7 downto 0);
V2_VALUE_F2:IN  std_logic_vector(7 downto 0);
V1_VALUE_F3:IN  std_logic_vector(7 downto 0);
V2_VALUE_F3:IN  std_logic_vector(7 downto 0);
V1_VALUE_F4:IN  std_logic_vector(7 downto 0);
V2_VALUE_F4:IN  std_logic_vector(7 downto 0);
--Enable_Machine:IN std_logic;
V1: OUT std_logic_vector(7 downto 0);
V2: OUT std_logic_vector(7 downto 0));
End component;
-------------------------------
Component  PWM_Generator  IS
port(Clock,Reset: IN std_logic;
new_data : in std_logic;
V1: IN std_logic_vector(7 downto 0) ;
V2 : IN std_logic_vector(7 downto 0) ;
PWM_V1 : OUT std_logic;
PWM_V2 : OUT std_logic);
End component;
Component  SPI_MASTER_2      IS
Generic (MAX_Time : Integer :=3; MAX_Temp_BITS: Integer :=16 ) ;
port(Clock,Reset,SDI: IN std_logic;
--MAX_Temp_BITS : IN integer range 0 to 255;
SCK: OUT std_logic;
SS : OUT std_logic;
TEMP: OUT std_logic_vector(15 downto 0));
End component;
```

```
Component    SPI_SLAVE    IS
Generic (MAX_BIT :integer := 136 ) ;
port(Clock,Reset,SDI,SCK,SS: IN std_logic;
ID: OUT std_logic_vector(47 downto 0);
V1_VALUE_F0:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F0:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F1:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F1:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F2:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F2:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F3:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F3:OUT  std_logic_vector(7 downto 0);
V1_VALUE_F4:OUT  std_logic_vector(7 downto 0);
V2_VALUE_F4:OUT  std_logic_vector(7 downto 0);
MAX_SLEEP_TIME: OUT std_logic_vector(7 downto 0) );
End component;
End My_components;
```

General1.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USe work.My_data_type.all;
USE work.my_components.all;
-------------------------------
ENTITY General1 IS
Generic (MAX_BIT : Integer :=48;MAX_Temp:integer:=16) ;
port(Clock,Reset                    : IN std_logic;
SDI_Master    : IN std_logic;
new_data                    : in std_logic;
SDI_Slave,SS_Slave,SCK_Slave      : in std_logic;
Osc_IN                      : IN std_Logic_vector(2 downto 0);
V1_sig,V2_sig                  : OUT std_logic;
SCK_Master,SS_Master            : OUT std_logic);
END General1;
-----------------------------------------------------
ARCHITECTURE behave OF General1 IS
SIGNAL   ID                         :  std_logic_vector(MAX_BIT-1 downto 0);
SIGNAL   MAX_sleep                  :  std_logic_vector(7 downto 0) ;
SIGNAL   Temperature_Out            :  std_logic_vector(MAX_Temp-1 downto 0) ;
SIGNAL   V1_VALUE_F0                      :  std_logic_vector(7 downto 0);
SIGNAL   V2_VALUE_F0                        :  std_logic_vector(7 downto 0);
SIGNAL   V1_VALUE_F1                      :  std_logic_vector(7 downto 0);
SIGNAL   V2_VALUE_F1                        :  std_logic_vector(7 downto 0);
SIGNAL   V1_VALUE_F2                        :  std_logic_vector(7 downto 0);
SIGNAL   V2_VALUE_F2                        :  std_logic_vector(7 downto 0);
SIGNAL   V1_VALUE_F3                        :  std_logic_vector(7 downto 0);
SIGNAL   V2_VALUE_F3                        :  std_logic_vector(7 downto 0);
SIGNAL   V1_VALUE_F4                        :  std_logic_vector(7 downto 0);
SIGNAL   V2_VALUE_F4                      :  std_logic_vector(7 downto 0);
```

```
SIGNAL              PWM_V1,PWM_V2                    :  std_logic_vector(7 downto 0);
SIGNAL   Osc_detect      :  frequency;
begin
Osc_detect <=   F0 when Osc_IN="000" ELSE
F1 when Osc_IN="001" ELSE
F2 when Osc_IN="010" ELSE
F3 when Osc_IN="011" ELSE
F4 when Osc_IN="100" ELSE
F4;
U1: COMPONENT Machine_state PORT MAP (Clock,Reset,new_data,ID,MAX_sleep,Temperature_Out,
Osc_detect,V1_VALUE_F0,V2_VALUE_F0,V1_VALUE_F1,V2_VALUE_F1,V1_VALUE_F2,V2_VALUE_F2,
V1_VALUE_F3,V2_VALUE_F3,V1_VALUE_F4,V2_VALUE_F4,PWM_V1,PWM_V2 );
U2: COMPONENT PWM_Generator PORT MAP  (Clock,Reset,new_data,PWM_V1,PWM_V2,V1_sig,V2_sig);
U4: COMPONENT SPI_MASTER_2  PORT MAP  (Clock,Reset,SDI_Master,SCK_Master,
SS_Master,Temperature_Out );
U6: COMPONENT SPI_SLAVE     PORT MAP  ( Clock,Reset,SDI_Slave,SCK_Slave,
SS_Slave,ID,V1_VALUE_F0,V2_VALUE_F0,V1_VALUE_F1,V2_VALUE_F1,
V1_VALUE_F2,V2_VALUE_F2,V1_VALUE_F3,V2_VALUE_F3,V1_VALUE_F4,V2_VALUE_F4,
MAX_sleep);
end behave;
```

SPI_Master_FSL_Test.vhd

```
---MODE 0
---At CPOL=0 the base value of the clock is zero
--For CPHA=0, data are read on the clock's rising edge
--and data are changed on a falling edge .
---- READ diffrent parameter from SPI and store it and write it back to state machine with
---- output port
---it is 17 bytes --- --- 6 bytes ID
---10 bytes V1,V2 values
---Defien maximum sleep time
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------------------------------
ENTITY SPI_Master_FSL_Test IS
Generic (MAX_BIT :integer := 16  ) ;
port(Clock,Reset,SCK,SS: IN std_logic;
Temperature:In Std_logic_vector(MAX_BIT-1 downto 0);
SDO:OUT std_logic);
END SPI_Master_FSL_Test;
ARCHITECTURE behave OF SPI_Master_FSL_Test IS
signal toggle_SCK:std_logic;
BEGIN
Process(Reset,Clock)
variable cnt_bits:integer range 0 to MAX_BIT-1;
BEGIN
if(Reset='1')then
cnt_bits:=0;--- number of bit recieve store
toggle_SCK<='1';  ---- initialization for register which store data
```

```
SDO<='0';
ELSIF(Clock'event and Clock='1' )then
if(ss='0')then  ---start id SS='0'
SDO<=Temperature(max_bit-cnt_bits-1); --store data on sdi
if(SCK='0' and toggle_SCK='1')then  ---write on falling edge
if(cnt_bits=MAX_BIT-1)then --- if all 152 receive issue data ready
cnt_bits:=0;
end if;
cnt_bits:=cnt_bits+1;
end if;
else  ---- if SS='1' initialize regsiters
cnt_bits:=0;
toggle_SCK<='1';
end if;
---- every time store clock data to detect rising and falling edge
if(SCK='1')then
toggle_SCK<='1';
else
toggle_SCK<='0';
end if;
end if;
END process;
END behave;
```

SPI_Slave_FSL_Test.vhd

```
--Data is clocked out ,on the falling edge of the serial clock (SC),
--Data is clocked in ,on the rising edge of SC.
----This is the  master spi for Test In board with FSL
-------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------------
ENTITY SPI_Slave_FSL_Test IS
Generic (MAX_BITs : Integer :=136 ; Max_Time: integer :=10) ;
port(Clock,Reset: IN std_logic;
FSL_IN_data:IN std_logic_vector(MAX_BITs-1 downto 0);
finish_SPI:out std_logic;
SDO: OUT std_logic;
SCK: OUT std_logic;
SS : OUT std_logic);
END SPI_Slave_FSL_Test;
-------------------------------------------------------------
ARCHITECTURE behave OF SPI_Slave_FSL_Test IS
Signal start:std_logic;
BEGIN
Process(Reset,Clock)
variable Cnt_CLk:integer  range 0 to (Max_Time*2+1);   --- counter to generate SCK
Variable Cnt_Bits:integer range 0 to MAX_BITs; ---- maximum bits to recive
variable Tmp_SCK:std_logic;---- Save previous generated clock
```

```
variable Counter_Byte:integer range 0 to 10; --- counter if it reach a bye or not
variable extra_delay:std_logic;  ---- rgister to define extra delay between bytes
variable W_FSL_IN_data: std_logic_vector(MAX_BITs-1 downto 0);
begin
if(Reset='1')then
---initialize all reg
SCK<='0';
SDO<='0';
SS<='1';
Cnt_Clk:=0;
start<='0';
Counter_Byte:=0;
Cnt_Bits:=0;
extra_delay:='0';
finish_SPI<='0';
Tmp_SCK:='0';
W_FSL_IN_data:=(others=>'0');
ELSIF(Clock'event and Clock='1'  )then
Cnt_Clk:=Cnt_Clk+1;
if(W_FSL_IN_data /= FSL_IN_data)then
finish_SPI<='0';
Start<='0';
Cnt_Clk:=0;
Cnt_Bits:=0;
Counter_Byte:=0;
SS<='1';
end if;
W_FSL_IN_data:=FSL_IN_data;
if( Start='0' and Cnt_clk=Max_Time*2 )then  --- wait for 2 at least time after each receive for settle time
Start<='1'; ---- start to recieve
ss<='0';---put the start signal for sensor
Cnt_Clk:=0;--- start to get times
Counter_Byte:=0;
elsif(start='1' and Cnt_clk=Max_Time  and Counter_Byte<8 and Cnt_Bits<MAX_BITs)then
SS<='0';
Cnt_Clk:=0;--set for another bit
if(Tmp_SCK='1')then
Tmp_SCK:='0'; ---Falling __data change on falling edge
else              ---RISING  __data read on rising edge
Tmp_SCK:='1';
SDO<=FSL_IN_data(MAX_BITs-Cnt_Bits-1); --data read from sensor
Cnt_Bits:=Cnt_Bits+1;
Counter_Byte:=Counter_Byte+1;
end if;
extra_delay:='0';  ---no delay , just after each recieve because of
SCK<=Tmp_SCK;  ----set CLOCK for spi
elsif(start='1' and Cnt_clk=Max_Time and Counter_Byte=8 and extra_delay='0')then
---- a byte recieve so add extra max_time delay
extra_delay:='1';
SDO<='0';
Tmp_SCK:='0';
```

```
SCK<='0';
Cnt_Clk:=0;
elsif(start='1' and Cnt_clk=Max_Time and Counter_Byte=8 and extra_delay='1')then
extra_delay:='0';  ----extra delay employed , back to recive next one
Cnt_Clk:=0;
Counter_Byte:=0;
SCK<='0';
elsif(start='1' and Cnt_clk=Max_Time and Cnt_Bits=MAX_BITs)then
----all bits receive make reg ready for next one
Cnt_Bits:=0;
Counter_Byte:=0;
SS<='1';
start<='0';
Cnt_Clk:=0;
finish_SPI<='1';
end if;
if(Cnt_Clk = Max_Time*2+1 )then
Cnt_Clk:=0;
end if;
end if;
end process;
end behave;
```

TEST_FSL_PWM.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
------------------------------------------------------------
ENTITY Test_FSL_PWM IS
Generic (MAX_Time : Integer :=256) ;
port(Clock,Reset: IN std_logic;
new_data                : in std_logic;
finish_PWM:out std_logic;
PWM_V1 : IN std_logic;
PWM_V2 : IN std_logic;
V1: out std_logic_vector(7 downto 0) ;
V2 : out std_logic_vector(7 downto 0) );
END Test_FSL_PWM;
------------------------------------------------------------
ARCHITECTURE behave OF Test_FSL_PWM IS
BEGIN
Process(Reset,Clock)
variable Counter_zero2,Counter_zero1:integer range 0 to MAX_Time+1;
variable Cnt_PWM_1,Cnt_PWM_2:integer range 0 to MAX_Time-1;
variable PWM_V1_tmp,PWM_V2_tmp:std_logic;
variable first_new_data:std_logic;
begin
```

```
if(Reset='1')then
V1<=(others=>'0');
V2<=(others=>'0');
PWM_V1_tmp:='0';
PWM_V2_tmp:='0';
Cnt_PWM_1:=0;
Cnt_PWM_2:=0;
Counter_zero1:=0;
Counter_zero2:=0;
finish_PWM<='0';
first_new_data:='0';
ELSIF(Clock'event and Clock='1'  )then
if(new_data='1')then
Cnt_PWM_1:=0;
Cnt_PWM_2:=0;
Counter_zero1:=0;
Counter_zero2:=0;
finish_PWM<='0';
end if;
Counter_zero1:=Counter_zero1+1;
Counter_zero2:=Counter_zero2+1;
if(PWM_V1='1')then
Cnt_PWM_1:=Cnt_PWM_1+1;
end if;--
if(PWM_V2='1')then
Cnt_PWM_2:=Cnt_PWM_2+1;
end if;
if( Counter_zero1=MAX_Time+1)then


V1 <=conv_std_logic_vector(Cnt_PWM_1,8);
V2 <=conv_std_logic_vector(Cnt_PWM_2,8);
Cnt_PWM_1:=0;
Cnt_PWM_2:=0;
finish_PWM<='1';
end if;
if(Counter_zero2=MAX_Time+1)then
Counter_zero2:=0;
end if;
if(Counter_zero1=MAX_Time+1)then
Counter_zero1:=0;
end if;
PWM_V1_tmp:=PWM_V1;
PWM_V2_tmp:=PWM_V2;
end if;
end process;
end behave;
```

my_Component_2.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USe work.My_data_type.all;
---------------------------------
Package My_component_2 Is
---------------------------------
component General1 IS
Generic (MAX_BIT : Integer :=48;MAX_Temp:integer:=16) ;
port(Clock,Reset                    : IN std_logic;
SDI_Master    : IN std_logic;
new_data                    : in std_logic;
SDI_Slave,SS_Slave,SCK_Slave       : in std_logic;
Osc_IN                      : IN std_Logic_vector(2 downto 0);
V1_sig,V2_sig                  : OUT std_logic;
SCK_Master,SS_Master             : OUT std_logic);
END component;
---------------------------------------
component Test_FSL_PWM IS
Generic (MAX_Time : Integer :=256) ;
port(Clock,Reset: IN std_logic;
new_data                : in std_logic;
finish_PWM:out std_logic;
PWM_V1 : IN std_logic;
PWM_V2 : IN std_logic;
V1: out std_logic_vector(7 downto 0) ;
V2 : out std_logic_vector(7 downto 0) );
END component;
---------------------------------------
Component SPI_Slave_FSL_Test IS
Generic (MAX_BITs : Integer :=136 ; Max_Time: integer :=10) ;
port(Clock,Reset: IN std_logic;
FSL_IN_data:IN std_logic_vector(MAX_BITs-1 downto 0);
finish_SPI:out std_logic;
SDO: OUT std_logic;
SCK: OUT std_logic;
SS : OUT std_logic);
End component;
-----------------------------------------------------------------
Component SPI_MASTER_FSL_Test IS
Generic (MAX_BIT :integer := 16  ) ;
port(Clock,Reset,SCK,SS: IN std_logic;
Temperature:In Std_logic_vector(MAX_BIT-1 downto 0);
SDO:OUT std_logic);
End component;
---------------------------------
End My_component_2;
```

General.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USe work.My_data_type.all;
USE work.my_component_2.all;
-------------------------------
ENTITY General IS
Generic (MAX_BIT : Integer :=48;MAX_Temp:integer:=16) ;
port(Clock,Reset               : IN std_logic;
finish_SPI                                                              : out std_logic;
new_data                 : in std_logic;
finish_PWM:out std_logic;
Osc_IN                   : IN std_Logic_vector(2 downto 0);
data_in          :in std_logic_vector(135 downto 0);
Temp_in          : in std_logic_vector(15 downto 0);
FSL_write                                                               :out std_logic_vector(15 downto 0) );
END General;
-----------------------------------------------------
ARCHITECTURE behave OF General IS
signal     V1_sig,V2_sig                :           std_logic;
signal   SCK_Master        :  std_logic;
signal   SS_Master          :  std_logic;
signal   SDI_Slave          :  std_logic;
signal   SCK_Slave          :  std_logic;
signal   SS_Slave           :  std_logic;
signal   SDI_Master         :  std_logic;
signal   W_V1               : std_logic_vector(7 downto 0);
signal   W_V2               : std_logic_vector(7 downto 0);
begin
FSL_write<=W_V1 & W_v2;
z1: COMPONENT Test_FSL_PWM  PORT MAP
(clock,Reset,new_data,finish_PWM,V1_sig,V2_sig,W_V1,W_V2);
Z2: Component General1 port map
(clock,Reset,SDI_Master,new_data,SDI_Slave,SS_Slave,SCK_Slave,Osc_IN,V1_sig,V2_sig,SCK_Master,SS_Mast
er);
z3: COMPONENT SPI_MASTER_FSL_Test  PORT MAP (clock,Reset,SCK_Master,SS_Master,
Temp_in,SDI_Master);
----1064
z4: COMPONENT SPI_Slave_FSL_Test  PORT MAP (clock,Reset,
data_in,finish_SPI,SDI_Slave,SCK_Slave,SS_Slave);
end behave;
```

General_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use ieee.std_logic_arith.all;
use work.My_data_type.all;
entity General_tb is
```

```
end General_tb;
architecture BEH of General_tb is
component General IS
Generic (MAX_BIT : Integer :=48;MAX_Temp:integer:=16) ;
port(Clock,Reset              : IN std_logic;
finish_SPI                                                    : out std_logic;
new_data                : in std_logic;
finish_PWM:out std_logic;
Osc_IN                  : IN std_Logic_vector(2 downto 0);
data_in          :in std_logic_vector(135 downto 0);
Temp_in          : in std_logic_vector(15 downto 0);
FSL_write                                                     :out std_logic_vector(15 downto 0) );
END component;
function str_to_stdvec(inp: string) return std_logic_vector is  ---the function convert string to std_vector
variable temp: std_logic_vector(inp'range) := (others => 'X');
begin
for i in inp'range loop
if (inp(i) = '1') then  ----character one
temp(i) := '1';    -----std logic '1'
elsif (inp(i) = '0') then
temp(i) := '0';
end if;
end loop;
return temp;
end function str_to_stdvec;
function stdvec_to_str(inp: std_logic_vector) return string is    ---convert std logic to str
variable temp: string(inp'left+1 downto 1) := (others => 'X');
begin
for i in inp'reverse_range loop
if (inp(i) = '1') then
temp(i+1) := '1';
elsif (inp(i) = '0') then
temp(i+1) := '0';
end if;
end loop;


return temp;
end function stdvec_to_str;
constant PERIOD : time := 10 ns;
signal W_Clock         : std_logic  := '0';   ----signal map to ports
signal W_RESET         : std_logic ;
signal    W_finish_SPI                                        :  std_logic;
signal    W_new_data,W_finish_PWM                                    :  std_logic;
signal    W_Osc_IN                :  std_Logic_vector(2 downto 0);
signal    W_data_in          : std_logic_vector(135 downto 0);
signal   W_Temp_in          :  std_logic_vector(15 downto 0);
signal    W_FSL_write                                         : std_logic_vector(15 downto 0) ;
signal  FSL_write_read    :  std_logic_vector(15 downto 0) ;
signal nt_first:std_logic;
file Osc_File: text open Read_mode is "Osc_new.txt";   -----contain F0 V1 V2
```

```
file ID_File: text open Read_mode is "ID_new.txt";     -----contain TAG ID
-----nothing to do
file Output_File: text open write_mode is "Outputs.txt";  ------ write result of every id in its exact position
begin
DUT : General   ------map ports
port map(Clock=>W_Clock,
Reset=> W_Reset,
finish_SPI          =>W_finish_SPI,
new_data            =>W_new_data,
finish_PWM=>w_finish_PWM,
Osc_IN    =>W_Osc_IN,
data_in   => W_data_in,
Temp_in  =>W_Temp_in,
FSL_write=>W_FSL_write);
W_Clock <= not W_Clock after PERIOD/2;   ----clock toggle every period/2
STIMULI : process
VARIABLE errors : BOOLEAN := false;
variable line_out,line_In1,line_In2: line; ---for line reading
Variable str_data_in : string  (136 downto 1 );---id read from file
Variable str_temp : string  (17 downto 1 );---id read from file
Variable Int_OSC : integer;
Variable Int_V1,Int_V2:integer;
Variable ID_data_in : std_logic_vector  (135 downto 0 );
variable TEMP_TMP:std_logic_vector  (15 downto 0 );
variable out_v1: STD_LOGIC_VECTOR(7 DOWNTO 0);
variable out_v2: STD_LOGIC_VECTOR(7 DOWNTO 0);
variable in_string:boolean;
begin
W_RESET<= '1';        ----reset system
W_new_data<='0';
wait for 20 ns ;
W_RESET<= '0';        ----toggle rest to start
wait for  PERIOD ;
while not (ENDFILE(ID_File)) loop   ---- read whole ID in order to get to the end of the file
readline (ID_File , line_In1);   ----read first line of file and save it to register
read (line_In1 , str_data_in);       ------put the line into a variale
read (line_In1 , str_temp);
--assert (false)
--       report "Vector: " & str_ID_TMP

--severity note;
ID_data_in := str_to_stdvec (str_data_in) ;       ----convert string to std_logic
TEMP_TMP:=str_to_stdvec (str_temp(16 downto 1)) ;
W_data_in <= ID_data_in;   --X"80CE64EEAD54";         ----put ID to port
W_Temp_in<= TEMP_TMP;
wait for  PERIOD*1000 ;
readline (Osc_File , line_In2);              ---read a line from the OSCillator
--wait for  PERIOD ;
in_string :=true;
while (W_finish_spi='0') loop
wait for  PERIOD/2;
end loop;
```

```
nt_first<='1';
while (in_string) loop
read (line_In2 , Int_OSC,in_string);  ----put read frequency to reg F0-F4
if not in_string then -- found end of line
exit;
end if;
W_new_data<='1';
read (line_In2 , Int_V1);   --- space now read V1
read (line_In2 , Int_V2);  ----space now read V2
out_v1:=CONV_STD_LOGIC_VECTOR (Int_V1,8);  ---convert them to std_logic
out_v2:=CONV_STD_LOGIC_VECTOR (Int_V2,8);
W_Osc_IN <= CONV_STD_LOGIC_VECTOR(Int_OSC,3);    ---put data to port
wait for period;
FSL_write_read <=Out_V1 & out_v2;
W_new_data<='0';
nt_first<='0';
--wait for period*254;
while(w_finish_PWM='0')loop
wait for period;
end loop;
IF (W_fsl_write /= FSL_write_read  )  THEN    ----see if output is diffrent from expected V1 in OSC file
ASSERT false         -- assert reports on false
REPORT "W_fsl_write is wrong";
errors := true;
END IF;
wait for period;
END LOOP;
wait for 1000 ns;
if( errors=true )then    -----if there is porblem report on screen and into file
write(line_out,string'("Test vector failed."));
writeline(Output_File ,line_out);
else
write(line_out,string'("Test vector passed!"));
writeline(Output_File ,line_out);
end if;
End loop;
---close the files
file_close(Output_File);
file_close(Osc_File);
file_close(ID_File);
-----write on the screen


ASSERT NOT errors
REPORT "Test vectors failed."
SEVERITY failure;
ASSERT errors
REPORT "Test vectors passed!"
SEVERITY failure;
WAIT;
end process STIMULI;
end BEH;
```

fsl.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity fsl is
port
(
FSL_Clk            : in        std_logic;
FSL_Rst: in        std_logic;
finish_SPI :in std_logic;
new_data:out std_logic;
finish_PWM:in std_logic;
bits_OSC    :out  std_logic_vector(2 downto 0);
FSL_write_to   :in std_logic_vector(15 downto 0);
FSL_S_Clk          : out       std_logic;
FSL_S_Read         : out       std_logic;
FSL_S_Data         : in        std_logic_vector(0 to 31);
FSL_S_Control      : in        std_logic;
FSL_S_Exists       : in        std_logic;
FSL_M_Clk          : out       std_logic;
FSL_M_Write        : out       std_logic;
FSL_M_Data         : out       std_logic_vector(0 to 31);
FSL_M_Control      : out       std_logic;
FSL_M_Full         : in        std_logic
);
attribute SIGIS : string;
attribute SIGIS of FSL_Clk : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";
end fsl;
architecture EXAMPLE of fsl is
signal start_read,Read_flag: std_logic;
signal out1:std_logic_vector(2 downto 0);
begin
FSL_M_Write <= not FSL_M_Full when  start_read='1' else '0';
The_SW_accelerator : process (FSL_Clk) is
variable tmp:std_logic_vector(31 downto 0);
begin  -- process The_SW_accelerator
if FSL_Clk'event and FSL_Clk = '1' then     -- Rising clock edge


if FSL_Rst = '1' then               -- Synchronous reset (active high)
Read_flag<='0';
new_data<='0';
start_read<='0';
tmp:=(others=>'0');
else
if(Finish_SPI='1')then
if(FSL_S_Exists = '1' and Read_flag='0')then
tmp:= std_logic_vector(unsigned(FSL_S_Data));
```

```
bits_OSC<=tmp(2 downto 0);
out1<=tmp(2 downto 0);
new_data<='1';
FSL_S_Read<='1';
Read_flag<='1';
else
new_data<='0';
FSL_S_Read<='0';
end if;
if(Read_flag='1' and finish_PWM='1')then
FSL_M_Data <= X"000" & '0'& out1 & FSL_write_to;
Read_flag<='0';
start_read<='1';
else
start_read<='0';
end if;
end if;
end if;
end if;
end process The_SW_accelerator;
end architecture EXAMPLE;
```

fsl_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FSL_test is
port
(
FSL_Clk          : in       std_logic;
FSL_Rst: in       std_logic;
FSL_S_Clk        : out      std_logic;
FSL_S_Read       : out      std_logic;
FSL_S_Data       : in       std_logic_vector(0 to 31);
FSL_S_Control    : in       std_logic;
FSL_S_Exists     : in       std_logic;
FSL_M_Clk        : out      std_logic;
FSL_M_Write      : out      std_logic;
FSL_M_Data       : out      std_logic_vector(0 to 31);
FSL_M_Control    : out      std_logic;
FSL_M_Full       : in       std_logic
);

end FSL_test;
architecture Behavioral of FSL_test is
------------------------------------
Component fsl is
port
(
FSL_Clk          : in       std_logic;
```

```
FSL_Rst: in         std_logic;
finish_SPI :in std_logic;
new_data:out std_logic;
finish_PWM:in std_logic;
bits_OSC    :out  std_logic_vector(2 downto 0);
FSL_write_to   :in std_logic_vector(15 downto 0);
FSL_S_Clk         : out       std_logic;
FSL_S_Read        : out       std_logic;
FSL_S_Data        : in        std_logic_vector(0 to 31);
FSL_S_Control     : in        std_logic;
FSL_S_Exists      : in        std_logic;
FSL_M_Clk         : out       std_logic;
FSL_M_Write       : out       std_logic;
FSL_M_Data        : out       std_logic_vector(0 to 31);
FSL_M_Control     : out       std_logic;
FSL_M_Full        : in        std_logic
);
end Component;
-----------------------------------------------------
Component General IS
Generic (MAX_BIT : Integer :=48;MAX_Temp:integer:=16) ;
port(Clock,Reset                : IN std_logic;
finish_SPI                                                          : out std_logic;
new_data                  : in std_logic;
finish_PWM:out std_logic;
Osc_IN                    : IN std_Logic_vector(2 downto 0);
FSL_write                                                           :out std_logic_vector(15 downto 0) );
END Component;
-------------------------------------------------------------------
Signal FSL_write   :std_logic_vector(15 downto 0);
Signal bits_OSC :std_logic_vector(2 downto 0);
signal finish_SPI:std_logic;
signal New_data:std_logic;
signal finish_PWM: std_logic;
begin
Z1:Component General  port map(FSL_Clk,FSL_Rst,finish_SPI,New_data,finish_PWM,bits_OSC,FSL_write);
Z2:Component fsl port map(FSL_Clk,FSL_Rst,finish_SPI,New_data,finish_PWM,bits_OSC,
FSL_write,FSL_S_Clk,FSL_S_Read,FSL_S_Data,FSL_S_Control,
FSL_S_Exists,FSL_M_Clk,FSL_M_Write,FSL_M_Data,FSL_M_Control,FSL_M_Full);
End Behavioral;
```

Appendix B

Source code of Test vector Generators

MSWOT

```
Private Sub Command_Click()
  Dim m, n, x, y, z, c, h, count, b, l, v
  Dim data(48)
  Open Path & "TAG_ID.txt" For Output As #1
  Open Path & "F0F4.txt" For Output As #2
  Randomize Timer
  For x = 1 To 50
    m = Int(Rnd * 16777215)
    n = Int(Rnd * 10000000)
    y = DecimalToBinary(m)
    c = DecimalToBinary(n)
    h = y & c
    lnglen = Len(h)
    count = 48 - lnglen
    For b = 1 To count
      h = Int(Rnd) & h
    Next b
    Print #1, h
    '-------------------------------------------------------------------------------------
    For t = 1 To 47  'Step 3
      strSubstr = Mid$(h, t, 1) & Mid$(h, t + 1, 1)
      If strSubstr = "00" Then
        data(t) = "f1 00 00"
      Else
        If strSubstr = "01" Then
          data(t) = "f3 01 01"
        Else
          If strSubstr = "10" Then
            data(t) = "f2 10 10"
          Else
            If strSubstr = "11" Then
              data(t) = "f4 11 11"
            End If
          End If
        End If
      End If
    Next t
    l = "0"
    For v = 1 To 48
      l = l & Space(1) & data(v)
    Next v
    Print #2, l
    '-------------------------------------------------------------------------------------
  Next x
  Close
```

```
MsgBox (" Tag IDs and Frequencies are generated and saved on TAG_ID.TXT and F0F4.TXT ")
End Sub
'-------------------------------------------------------------------------------------
Public Function DecimalToBinary(DecimalNum) As String


  Dim tmp As String
  Dim n As Long
  n = DecimalNum
  tmp = Trim(Str(n Mod 2))
  n = n \ 2
  Do While n <> 0
    tmp = Trim(Str(n Mod 2)) & tmp
    n = n \ 2
  Loop
  DecimalToBinary = tmp
End Function
```

T.V.Gen.

```
Private Sub Command_Click()
  Command.Caption = "Please wait ..."
  Command.Enabled = False
  Text1.Enabled = False
  Text2.Enabled = False
  Text3.Enabled = False
  Text4.Enabled = False
  Text5.Enabled = False
  Dim m, n, x, y, z, c, h, count, b, l, v, a1, TON, tempdata, v2, l2, error_tmp
  Dim data(57)
  Dim rnd_no, initial_count As Integer
  Dim temp_all, temp_data As String
  Open Path & "ID.txt" For Output As #1
  Open Path & "Osc.txt" For Output As #2
  error_tmp = "1 1 E 1 1"
  Randomize Timer
  For x = 1 To Text1.Text
    m = Int(Rnd * 16777215)
    n = Int(Rnd * 10000000)
    y = DecimalToBinary(m)
    c = DecimalToBinary(n)
    h = y & c
    lnglen = Len(h)
    count = 48 - lnglen
    For b = 1 To count
      h = Int(Rnd) & h
    Next b
    crnd = Int(Rnd * 100)
    ctemp2 = DecimalToBinary(crnd)
```

```
ctemp3 = Len(ctemp2)
   count_temp = 8 - ctemp3
   For btemp = 1 To count_temp
     ctemp2 = Int(Rnd) & ctemp2
   Next btemp
   vvv = h & Space(1) & ctemp2            'ID FROMAT
   Print #1, vvv


 For t = 1 To 47  'Step 2
     strSubstr = Mid$(h, t, 1) & Mid$(h, t + 1, 1)
     If strSubstr = "00" Then
       data(t) = Text2.Text
     Else
       If strSubstr = "10" Then
         data(t) = Text4.Text
       Else
         If strSubstr = "01" Then
           data(t) = Text3.Text
         Else
           If strSubstr = "11" Then
             data(t) = Text5.Text
           End If
         End If
       End If
     End If
   Next t
   If data(47) = "1 0 0" Then
    l = "0 17 17 1 0 0"
   Else
     If data(47) = "2 1 0" Then
       l = "0 17 17 2 1 0"
     Else
       If data(47) = "3 0 1" Then
         l = "0 17 17 3 0 1"
       Else
         If data(47) = "4 1 1" Then
           l = "0 17 17 4 1 1"
         End If
       End If
     End If
   End If
   For v = 47 To 1 Step -1
    l = l & Space(1) & data(v)
   Next v
   TON = (Int((4 * Rnd()) + 1))   ' Temp or No Temp
   If TON = 2 Or TON = 3 Or TON = 1 Then

     '------------------------- Temp DATA INTO OSC------------------

     For tempdata = 49 To 56     'Step 2
```

```
        strSubstr = Mid$(vvv, tempdata, 1) & Mid$(vvv, tempdata + 1, 1)
        If strSubstr = "00" Then


data(tempdata) = "0 0"
        Else
          If strSubstr = "10" Then
            data(tempdata) = "1 0"
          Else
            If strSubstr = "01" Then
              data(tempdata) = "0 1"

            Else

 If strSubstr = "11" Then
 data(tempdata) = "1 1"
              End If
            End If
          End If
        End If
      Next tempdata
      For v2 = 56 To 50 Step -1
        If data(v2) = "1 1" Then
          l2 = l2 & Space(1) & error_tmp & Space(1) & "2"
        Else
          l2 = l2 & Space(1) & data(v2) & Space(1) & "2"
        End If
      Next v2
      Print #2, l & Space(1) & "2 1 0 2" & l2 & Space(1) & "1 0"
      '------------------------End Temp Data Into OSC----------------------------
    Else
      Print #2, l & Space(1) & 3 & Space(1) & 255 & Space(1) & 255
    End If
    '-------------------------------------------------------------------------------------
    l2 = ""
  Next x
  Close
  MsgBox " Tag IDs and Frequencies are generated and saved on ID.txt and Osc.txt ", 0, "Done!"
  Command.Enabled = True
  Command.Caption = "Generate TAG IDs and Frequencies"
  Text1.Enabled = True
  Text2.Enabled = True
  Text3.Enabled = True
  Text4.Enabled = True
  Text5.Enabled = True
End Sub
'-------------------------------------------------------------------------------------
Public Function DecimalToBinary(DecimalNum) As String
  Dim tmp As String
  Dim n As Long
  n = DecimalNum
  tmp = Trim(Str(n Mod 2))
  n = n \ 2
```

```
  Do While n <> 0
    tmp = Trim(Str(n Mod 2)) & tmp
    n = n \ 2
  Loop


DecimalToBinary = tmp
End Function
```

PWMT

```
Private Sub Command_Click()
Command.Caption = "Please wait ..."
  Command.Enabled = False
  Text1.Enabled = False
  Dim m, n, x, y, c, h, z, h1, dem1, dem2, strSubstr, strSubstr2
  Open Path & "PWM_2.txt" For Output As #1
  Open Path & "Result_PWM.txt" For Output As #2
  Randomize Timer
  For x = 1 To Text1.Text
    m = Int(Rnd * 225)
    n = Int(Rnd * 225)
    y = DecimalToBinary(m)
    c = DecimalToBinary(n)
    Do Until Len(y) >= 8
      dem1 = DecimalToBinary(Int(Rnd * 10 / 8))
      y = dem1 & y
    Loop
    Do Until Len(c) >= 8
      dem2 = DecimalToBinary(Int(Rnd * 10 / 8))
      c = dem2 & c
    Loop
    h = y & Space(1) & c
    Print #1, h
    '-----------------------------------------------------------------------------------------
    strSubstr = Mid$(h, 1, 8)          ' 1st bin
    strSubstr2 = Mid$(h, 10, 8)        ' 2nd bin
    z = BinToDec(strSubstr) & Space(1) & BinToDec(strSubstr2)
    Print #2, z
  Next x
    Close
  MsgBox " All Data are generated and saved on PWM_2.txt and Result_PWM.txt ", 0, "Mazi"
  Command.Enabled = True
  Command.Caption = "Generate PWM"
  Text1.Enabled = True
End Sub
Public Function DecimalToBinary(DecimalNum) As String
  Dim tmp As String
  Dim n As Long
  n = DecimalNum
  tmp = Trim(Str(n Mod 2))
```

```
n = n \ 2
  Do While n <> 0
    tmp = Trim(Str(n Mod 2)) & tmp
    n = n \ 2
  Loop
  DecimalToBinary = tmp
End Function
Public Function BinToDec(Binary) As String
  Dim BinaryNum
  Dim BitCount
  For BitCount = 1 To Len(Binary)


    BinaryNum = BinaryNum + (CDbl(Mid(Binary, Len(Binary) - BitCount + 1, 1)) * (2 ^ (BitCount - 1)))
  Next BitCount
  BinToDec = BinaryNum
End Function
```

MFTVG

```
Private Sub Command_Click()
  Command.Caption = "Please wait ..."
  Command.Enabled = False
  Text1.Enabled = False
  Text2.Enabled = False
  Text7.Enabled = False
  Text3.Enabled = False
  Text8.Enabled = False
  Text4(0).Enabled = False
  Text9.Enabled = False
  Text5(1).Enabled = False
  Text10.Enabled = False
  Text6.Enabled = False
  Text11.Enabled = False
  Text12.Enabled = False
  Dim m, n, x, y, z, c, h, count, b, l, v, a1, TON, tempdata, v2, l2, error_tmp
  Dim data(200)
  Dim rnd_no, initial_count As Integer
  Dim temp_all, temp_data As String
  Open Path & "Test1.txt" For Output As #1
  Open Path & "Test_result1.txt" For Output As #2
  error_tmp = "1 1 E 1 1"
  Randomize Timer
  For x = 1 To Text1.Text
    m = Int(Rnd * 16777215)
    n = Int(Rnd * 10000000)
    y = DecimalToBinary(m)
    c = DecimalToBinary(n)
    h = y & c
```

```
 lnglen = Len(h)
   count = 48 - lnglen
   For b = 1 To count
     h = Int(Rnd) & h
   Next b
   crnd = Int(Rnd * 100)
   ctemp2 = DecimalToBinary(crnd)
   ctemp3 = Len(ctemp2)
   count_temp = 16 - ctemp3       'bit counter
   For btemp = 1 To count_temp
     ctemp2 = Int(Rnd) & ctemp2
   Next btemp
   vvv = h & Text2.Text & Text7.Text & Text3.Text & Text8.Text & Text4(0).Text & Text9.Text & Text5(1).Text &
Text10.Text & Text6.Text & Text11.Text & Space(1) & Text15.Text  'ctemp2               'ID FROMAT

Print #1, vvv
   '-------------------------------------------------------------------------------------
   For t = 1 To 47  'Step 2
     strSubstr = Mid$(h, t, 1) & Mid$(h, t + 1, 1)
     If strSubstr = "00" Then
       data(t) = "1" & Space(1) & BinToDec(Text3.Text) & Space(1) & BinToDec(Text8.Text)
     Else
       If strSubstr = "10" Then
         data(t) = "3" & Space(1) & BinToDec(Text5(1).Text) & Space(1) & BinToDec(Text10.Text)
       Else
         If strSubstr = "01" Then
           data(t) = "2" & Space(1) & BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text)
         Else
           If strSubstr = "11" Then
             data(t) = "4" & Space(1) & BinToDec(Text6.Text) & Space(1) & BinToDec(Text11.Text)
           End If
         End If
       End If
     End If
   Next t
   If data(47) = "1" & Space(1) & BinToDec(Text3.Text) & Space(1) & BinToDec(Text8.Text) Then
     l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "1" & Space(1)
& BinToDec(Text3.Text) & Space(1) & BinToDec(Text8.Text)
   Else
     If data(47) = "2" & Space(1) & BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text) Then
       l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "2" & Space(1)
& BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text)
     Else
       If data(47) = "3" & Space(1) & BinToDec(Text5(1).Text) & Space(1) & BinToDec(Text10.Text) Then
         l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "3" &
Space(1) & BinToDec(Text5(1).Text) & Space(1) & BinToDec(Text10.Text)
       Else
         If data(47) = "4" & Space(1) & BinToDec(Text6.Text) & Space(1) & BinToDec(Text11.Text) Then
           l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "4" &
Space(1) & BinToDec(Text6.Text) & Space(1) & BinToDec(Text11.Text)
         End If
       End If
```

```
      End If


End If
   For v = 47 To 1 Step -1
    l = l & Space(1) & data(v)
   Next v
   '------------------------- Temp DATA INTO OSC------------------
   For tempdata = 130 To 145  '112  'Step 2
    strSubstr = Mid$(vvv, tempdata, 1) & Mid$(vvv, tempdata + 1, 1)
    If strSubstr = "00" Then
     data(tempdata) = BinToDec(Text3.Text) & BinToDec(Text8.Text)
    Else
     If strSubstr = "10" Then
      data(tempdata) = BinToDec(Text4(0).Text) & BinToDec(Text9.Text)
     Else
      If strSubstr = "01" Then
       data(tempdata) = BinToDec(Text5(1).Text) & BinToDec(Text10.Text)


      Else
       If strSubstr = "11" Then
        data(tempdata) = BinToDec(Text6.Text) & BinToDec(Text11.Text)
       End If
      End If
     End If
    End If
   Next tempdata
   For v2 = 145 To 130 Step -1
    l2 = l2 & Space(1) & data(v2) & Space(1) & "2"
   Next v2
   Print #2, l & Space(1) & "2 1 0 2" & l2 & Space(1) & "1 0"
   l2 = ""
  Next x
  Close
  MsgBox " Requested data are generated and saved on Test1.txt and Test_result1.TXT ", 0, "Mazi"
  Command.Enabled = True
  Command.Caption = "Generate"
  Text1.Enabled = True
  Text2.Enabled = True
  Text7.Enabled = True
  Text3.Enabled = True
  Text8.Enabled = True
  Text4(0).Enabled = True
  Text9.Enabled = True
  Text5(1).Enabled = True
  Text10.Enabled = True
  Text6.Enabled = True
  Text11.Enabled = True
  Text12.Enabled = True
End Sub
Public Function DecimalToBinary(DecimalNum) As String
  Dim tmp As String
```

```
  Dim n As Long
  n = DecimalNum
  tmp = Trim(Str(n Mod 2))


 n = n \ 2
  Do While n <> 0
    tmp = Trim(Str(n Mod 2)) & tmp
    n = n \ 2
  Loop
  DecimalToBinary = tmp
End Function
Public Function BinToDec(Binary) As String
  Dim BinaryNum
  Dim BitCount
  For BitCount = 1 To Len(Binary)
    BinaryNum = BinaryNum + (CDbl(Mid(Binary, Len(Binary) - BitCount + 1, 1)) * (2 ^ (BitCount - 1)))
  Next BitCount
  BinToDec = BinaryNum
End Function
```

AFTVG

```
Private Sub Command_Click()
  Command.Caption = "Please wait ..."
  Command.Enabled = False
  Form1.Enabled = False
  Dim m, n, x, y, z, c, h, count, b, l, v, a1, TON, tempdata, v2, l2, error_tmp, count8, l3
  Dim data(200), arr(13)
  Dim rnd_no, initial_count As Integer
  Dim temp_all, temp_data As String
  Open Path & "ID_new.txt" For Output As #1
  Open Path & "Osc_new.txt" For Output As #2
  error_tmp = "1 1 E 1 1"
  Randomize Timer
  For x = 1 To Text1.Text
    m = Int(Rnd * 16777215)
    n = Int(Rnd * 10000000)
    y = DecimalToBinary(m)
    c = DecimalToBinary(n)
    h = y & c
    lnglen = Len(h)
    count = 48 - lnglen
    For b = 1 To count
      h = Int(Rnd) & h
    Next b
    For count8 = 1 To 13
      crnd = Int(Rnd * 100)
      ctemp2 = DecimalToBinary(crnd)
      ctemp3 = Len(ctemp2)
      count_temp = 8 - ctemp3       'bit counter
```

```
      For btemp = 1 To count_temp
        ctemp2 = Int(Rnd) & ctemp2
      Next btemp
      arr(count8) = ctemp2


Next count8
    Text2.Text = arr(1)
    Text7.Text = arr(2)
    Text3.Text = arr(3)
    Text8.Text = arr(4)
    Text4(0).Text = arr(5)
    Text9.Text = arr(6)
    Text5(1).Text = arr(7)
    Text10.Text = arr(8)
    Text6.Text = arr(9)
    Text11.Text = arr(10)
    Text12.Text = arr(11)
    Text15.Text = arr(12) & arr(13)
    vvv = h & Text2.Text & Text7.Text & Text3.Text & Text8.Text & Text4(0).Text & Text9.Text & Text5(1).Text &
Text10.Text & Text6.Text & Text11.Text & Text12.Text & Space(1) & Text15.Text              'ID FROMAT
    Print #1, vvv
    '-------------------------------------------------------------------------------------------
    For t = 1 To 47  'Step 2
      strSubstr = Mid$(h, t, 1) & Mid$(h, t + 1, 1)


      If strSubstr = "00" Then
        data(t) = "1" & Space(1) & BinToDec(Text3.Text) & Space(1) & BinToDec(Text8.Text)
      Else
        If strSubstr = "10" Then
          data(t) = "3" & Space(1) & BinToDec(Text5(1).Text) & Space(1) & BinToDec(Text10.Text)
        Else
          If strSubstr = "01" Then
            data(t) = "2" & Space(1) & BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text)
          Else
            If strSubstr = "11" Then
              data(t) = "4" & Space(1) & BinToDec(Text6.Text) & Space(1) & BinToDec(Text11.Text)
            End If
          End If
        End If
      End If
    Next t
    If data(47) = "1" & Space(1) & BinToDec(Text3.Text) & Space(1) & BinToDec(Text8.Text) Then
      l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "1" & Space(1)
& BinToDec(Text3.Text) & Space(1) & BinToDec(Text8.Text)
    Else
      If data(47) = "2" & Space(1) & BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text) Then
        l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "2" & Space(1)
& BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text)
      Else
        If data(47) = "3" & Space(1) & BinToDec(Text5(1).Text) & Space(1) & BinToDec(Text10.Text) Then
          l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "3" &
```

```
Space(1) & BinToDec(Text5(1).Text) & Space(1) & BinToDec(Text10.Text)
       Else
        If data(47) = "4" & Space(1) & BinToDec(Text6.Text) & Space(1) & BinToDec(Text11.Text) Then
         l = "0" & Space(1) & BinToDec(Text2.Text) & Space(1) & BinToDec(Text7.Text) & Space(1) & "4" &
Space(1) & BinToDec(Text6.Text) & Space(1) & BinToDec(Text11.Text)
        End If


End If
     End If
   End If
   For v = 47 To 1 Step -1
    l = l & Space(1) & data(v)
   Next v
   '------------------------ Temp DATA INTO OSC------------------
   For tempdata = 138 To 153  '112  'Step 2
    strSubstr = Mid$(vvv, tempdata, 1) & Mid$(vvv, tempdata + 1, 1)
    If strSubstr = "00" Then
     data(tempdata) = BinToDec(Text3.Text) & Space(1) & BinToDec(Text8.Text)
    Else
     If strSubstr = "10" Then
      data(tempdata) = BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text)
     Else
      If strSubstr = "01" Then
       data(tempdata) = BinToDec(Text5(1).Text) & Space(1) & BinToDec(Text10.Text)
      Else
       If strSubstr = "11" Then
        data(tempdata) = BinToDec(Text6.Text) & Space(1) & BinToDec(Text11.Text)
       End If


      End If
     End If
    End If
   Next tempdata
   For v2 = 153 To 138 Step -1
    l2 = l2 & Space(1) & data(v2) & Space(1) & "2"
   Next v2
   Print #2, l & Space(1) & "2" & Space(1) & BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text) & l2 &
Space(1) & "0 0"  'BinToDec(Text4(0).Text) & Space(1) & BinToDec(Text9.Text)

   '-----------------------End Temp Data Into OSC----------------------------
   l2 = ""
  Next x
  Close
  MsgBox " Requested data are generated and saved on ID_new.txt and Osc_new.txt ", 0, "Mazi"
  Command.Enabled = True
  Command.Caption = "Generate"
  Form1.Enabled = True
  Text2.Text = BinToDec(arr(1))
  Text7.Text = BinToDec(arr(2))
  Text3.Text = BinToDec(arr(3))
  Text8.Text = BinToDec(arr(4))
```

```
  Text4(0).Text = BinToDec(arr(5))
  Text9.Text = BinToDec(arr(6))
  Text5(1).Text = BinToDec(arr(7))
  Text10.Text = BinToDec(arr(8))
  Text6.Text = BinToDec(arr(9))
  Text11.Text = BinToDec(arr(10))
End Sub



Public Function DecimalToBinary(DecimalNum) As String
Dim tmp As String
  Dim n As Long
  n = DecimalNum
  tmp = Trim(Str(n Mod 2))
  n = n \ 2
  Do While n <> 0
    tmp = Trim(Str(n Mod 2)) & tmp
    n = n \ 2
  Loop
  DecimalToBinary = tmp
End Function
Public Function BinToDec(Binary) As String
  Dim BinaryNum
  Dim BitCount
  For BitCount = 1 To Len(Binary)
    BinaryNum = BinaryNum + (CDbl(Mid(Binary, Len(Binary) - BitCount + 1, 1)) * (2 ^ (BitCount - 1)))
  Next BitCount
  BinToDec = BinaryNum
End Function
```