Institut Universitaire Technologique Calais-Boulogne
Département Génie Electrique et Informatique Industrielle

Stage report

Spark Generator Board
&
Path Finder Robot

Stage from 14 April to 19 June 1998

Student : Stephane ODUL
Responsible teacher : Mr Nickema
Supervisor : Dr K.R. Dimond

ACKNOWLEDGMENT

First of all I would like to thank Dr K.R. Dimond, who welcomed me to the Electronics Laboratory during these 10 weeks.

I would also like to thanks Mr John Bevan, Mathematician at the Canterbury Business School who introduced me to Dr K. R. Dimond.

Moreover, I would like to thank Mr Harvey Twyman for showing me how the development tools worked, and for his technical assistance in my projects.

I would like to thank all the staff who provided me with the necessary for those projects.

Finally, I would like to thank Miss Emanuelle Andre who give me the opportunity to do my work placement in England.

CONTENTS

INTRODUCTION

At the end of our two year course we were required to carry out a work experiment lasting ten weeks to conclude our studies. I chose to do my experiment in England, in the .Electronic Engineering Laboratory of the university of Kent at Canterbury.

My experiment is based on the use of the VHDL language and consist of two main projects, a Spark Engine Generator and a Path Finder Robot.

At first I spent two weeks to be familiar with the VHDL, by completing a project based on the Pulse Position Modulation serial transmission.

The Spark Engine Generator can be used in car engines and provides a more efficient explosion of the petrol. This project takes 3 weeks to complete, and worked without any noticeable trouble.

The Path Finder Robot is a good introduction to robotics and stand-alone tools. After five weeks of development this project still have some possible necessary improvements.

I hope you enjoy reading my work which describes various steps of the project.

Presentation of the Electronic Laboratory of the University of Kent at Canterbury

The University was founded 35 years ago.

Electronic Engineering Laboratory founded 30 years ago.

The department has over 20 full-time academic staff, and approx 24 full-time research staff. Funded by research councils and industry.

The department offers a number of undergraduate courses, electronic engineering, communication engineering, electronics and medical electronics, electronic system engineering and computer system engineering. In addition there is a masters course in communication systems engineering.

Student numbers are approx 300 full-time undergraduates and 50 postgraduates, taught courses and research, and 8 000 students on the campus.

The Research Faculties are divided into two domains: communications and digital systems. Themes of research in communications include Antenna options, GaAs microwave circuits, Opto-electronics for networks, microwave measure units and radio astronomy. Research themes in Digital Systems are Image Processing, CAD for VLSI, Medical Instrumentation and Neural Systems.

Introduction to VHDL

1.1. Presentation of VHDL

VHDL is an acronym which stands for VHSIC Hardware Description Language. VHSIC is yet another acronym which stands for Very High Speed Integrated Circuits. The language has been known to be somewhat complicated, as its title (as titles go). The acronym does have a purpose, though; it is supposed to capture the entire theme of the language, that is to describe hardware much the same way we use schematics.

To make designs more understandable and maintainable, a design is typically decomposed into several blocks. These blocks are then connected together to form a complete design. Using the schematic capture approach to design, this might be done with a block diagram editor. Every portion of a VHDL design is considered a block. A VHDL design may be completely described in a single block, or it may be decomposed in several blocks. Each block in VHDL is analogous to an off-the-shelf part and is called an entity. The *entity* describes the interface to that block and a separate part associated with the entity describes how that block operates. The interface description is like a pin description in a data book, specifying the inputs and outputs to the block. The description of the operation of the part is like a schematic for the block.

```
ENTITY latch IS
  PORT (s,r  : IN  BIT;
        q,nq : OUT BIT);
END latch;
```

In this example the ENTITY statement declare a new component which name is latch, the PORT statement is used for the declaration of the input and output signal of the component. This component declaration can be assimilated to a function declaration in a high level language such as Pascal or C, but you can notice that a VHDL entity allow more than one output.

```
ARCHITECTURE dataflow OF latch IS
BEGIN
  q  <= r NOR nq;
  nq <= s NOR q;
END dataflow;
```

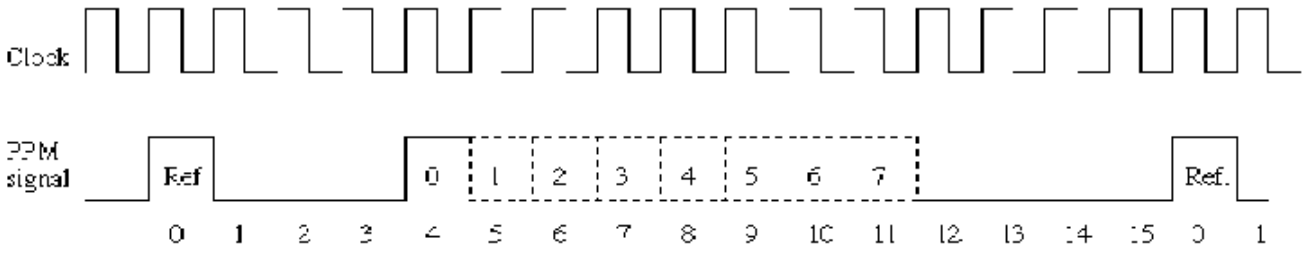
The second part of the description, the architecture declaration, is a description of how the component operates. The *dataflow* word is use as a description of the architecture, as far as you can have more then one possible architecture for the same component.

1.2. Example in VHDL : PPM emitter and PPM receiver

1.2.1. The PPM signal

PPM (Pulse Position Modulation) is a serial data transmission protocol.

The PPM signal is composed of a reference pulse followed, a few clock cycles later, by another pulse, the width of this second pulse gives the value sent by the emitter.



The advantage of the PPM signal is that the transmission is very reliable: due to the form of the signal an error can be detected very easily. This signal is by the way ideal for use in a very noisy environment.

The disadvantage is that the signal use a huge amount of band-width, in our example it's using 16 clock cycles for a 3 bit value, whereas others serial transmissions could use less than 6 clock cycles.

All this explain why the PPM transmission is chosen for most of the infra red remote controllers like television or CD-player remote controller.

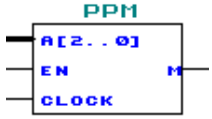
1.2.2. The PPM emitter

The PPM emitter receive 3 input signals: a 3 bit integer (the value to transmit), an enabler, and the clock. There is only one output which is the PPM signal itself.

This is declared very simply in the entity declaration of the PPM emitter:

```
ENTITY ppm IS
   PORT( a     : IN  BIT_VECTOR (2 DOWNTO 0);
         en    : IN  BIT;
         clock : IN  BIT;
         m     : OUT BIT);
END ppm;
```

Symbol of the ppm



The PPM signal is very simple to implement in VHDL, the process is only a 16 clock cycle loop which generates the reference pulse at first, and sets the signal to one at the fourth clock cycle, and finally reset it to zero when the input value, here b, is reached.

Here the value is stored in b which is an integer, whereas a is a vector of 3 bits. The VHDL language is very strict and does not allow you to operate tests between different types of signals or variables. That's why b is used here instead of a directly.

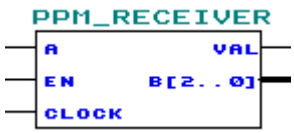
```
PROCESS (clock)
VARIABLE n : INTEGER RANGE 0 TO 15 := 0;
BEGIN
    IF clock = '1' THEN
        IF en = '1' THEN
            IF n = 15 THEN
                n := 0 ;
            ELSE
                IF      n = 0 THEN  m <= '1';  -- Reference
                ELSIF   n = 1 THEN  m <= '0';  -- Wait until 4
                ELSIF   n = 4 THEN  m <= '1';  -- Set to '1'
                ELSIF   n>4 THEN               -- Until a value
                    IF (n - 4) > b THEN
                        m <= '0';
                    END IF;
                END IF;
                n := n + 1;
            END IF;
        ELSE    -- en = '0'
            n := 0;
            m <= '0';
        END IF;
    END IF;
END PROCESS;
```

1.2.3. The PPM receiver

The PPM receiver is a little bit more complex than the emitter. This time the signal must be recognised instead of being created. The method used here to recognise the signal is to store the signal in a 16 bit memory, which is the length of the signal. This memory is compared to the possible formats of the signal, and when the signal match, we set the corresponding value to the output.

The PPM receiver have three input signal : the PPM signal itself, an enabler, and the clock. This time we use 2 outputs : a 3 bit integer which contain the result of the last match (the transmitted value), and a validation signal, this last signal is necessary because it announces when the signals match and so is useful to know if the output value

of the receiver is the value of the current signal or only the stored value of the last signal. It can be used too as a flag to check if the transmission occurs and is correct.




```
ENTITY ppm_receiver IS
    PORT(
        a       : IN STD_LOGIC;
        en      : IN STD_LOGIC;
        clock   : IN STD_LOGIC;
        val     : OUT STD_LOGIC;
        b       : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
    );
END ppm_receiver;
```

As previously said, we compare a stored signal to the possible formats. If the signal is not a possible signal, nothing occurs, but if it matches, the corresponding value is send to the output, and the val (validate) signal is set to one to announce that a PPM signal has been successfully received.

```
ARCHITECTURE stdlogic OF ppm_receiver IS
    SIGNAL mem : STD_LOGIC_VECTOR (0 TO 15) := "0000000000000000";
BEGIN
    PROCESS (clock)
    BEGIN
        IF clock = '1' THEN
            FOR i IN 0 TO 14 LOOP
                mem(i) <= mem(i+1);
            END LOOP;
            mem(15) <= a;
            IF en = '1' AND mem(0 TO 4) = "10001"
             AND mem (12 TO 15) = "0000" THEN
                CASE mem(5 TO 11) IS
                    WHEN "0000000" =>
                        val <= '1';
                        b   <= "000";
                    WHEN "1000000" =>
                        val <= '1';
                        b   <= "001";
                    . we show only the beginning and the
                    . end for space reasons
                    WHEN "1111111" =>
                        val <= '1';
                        b   <= "111";
                    WHEN OTHERS =>  val <= '0';
                END CASE;
            ELSE    -- en = '0'
                val <= '0';
            END IF;
        END IF;
    END PROCESS;
END stdlogic;
```

1.2.4. Linking the components together

Now we have two components : a PPM emitter and a PPM receiver, but they are useless if we cannot link them together.

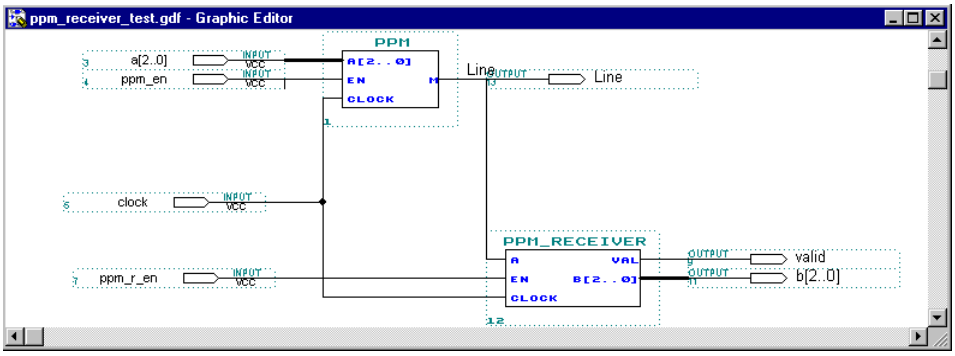
The VHDL allows us to use a VHDL component as a sub-component of another VHDL component. In order to use them we have to declare the component with their PORT assignment for the inputs and outputs. This is made just after the ARCHITECTURE statement.

```
ARCHITECTURE stdlogic OF ppm_fpga IS
COMPONENT ppm IS
    PORT(
        a, en, clock    : IN  BIT;
        m               : OUT BIT);
END COMPONENT;
FOR ALL : ppm USE ENTITY work.ppm(ver_max);
COMPONENT ppm_receiver IS
    PORT(
        a, en, clock    : IN STD_LOGIC;
        val             : OUT STD_LOGIC;
        b               : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END COMPONENT;
FOR ALL : ppm USE ENTITY work.ppm(ver_max);
```

After the declaration of the components, we don't use a process but set a label followed by the signal assignment to the ports of the components, i.e. the port mapping. Internal signals have to be declared to link the component together.

```
    SIGNAL  line, valid : STD_LOGIC;
    SIGNAL  a, b        : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    p0 : ppm PORT MAP(a, pba, clock, line);
    P1 : ppm_receiver PORT MAP(line, pbb, clock, valid,b);
```

The result is equivalent to the result we could obtain with a schematic editor.



1.3. Testing the PPM system

1.3.1. The Max+PlusII environment

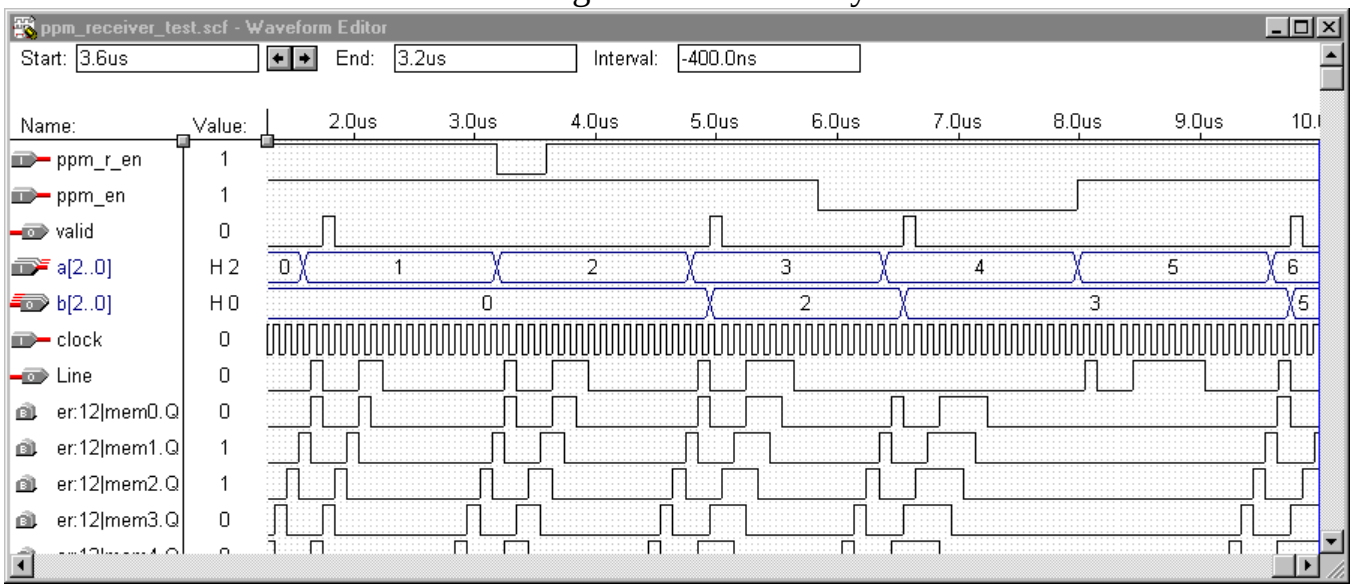
The VHDL, as any language, needs a compiler. the most popular VHDL compiler is the compiler provided in the Integrated Development Environment from Altera, called Max+PlusII. This environment supports many ways to develop hardware components, like AHDL (Altera Hardware Description Language), or a schematic editor.

The Max+PlusII software provide all the tools necessary for the development of the Programmable Gate Array (FPGA) chips. Those include a text editor, a schematic editor, the compiler, a waveform editor, a simulator, and a programmer to download the program into the chip.

1.3.2. Simulation of the PPM system

In order to simulate a VHDL component we have to create a waveform file. In this file we declare the value of the input signals, and declare the output signals we want to simulate. Then we start the simulation.

Simulated signals of the PPM system



The signal a[2..0] is the input value of the PPM emitter, and b[2..0] is the output value of the PPM receiver. The line signal is the PPM signal on the «line» between the emitter and the receiver.

Some of the memory bits of the PPM receiver are shown on this simulation. We can see how those bits are evolving during the time corresponding to the value of the PPM signal.

The valid signal indicate when the PPM receiver detects a «valid» signal. It is useful to differentiate a long signal from two signals with the same value.

1.3.3. The testing board

The Digital System Laboratory is equipped with a testing kit for each computer with the Max+PlusII software.

The testing kit

The kit is composed of an Altera board, a logic analyser, and an oscilloscope.

The Altera board has an Altera EPF8452ALC84 chip which is used for each project. The memory of this chip is volatile though it is necessary to use an external EPROM for stand alone projects.

The board has 4push buttons directly connected to the chip, 2 hexadecimal switches, and 2 hexadecimal displays. Those last components are accessed using a data bus.

In order to test the PPM system, we used one of the hexadecimal switches as the input signal for the PPM emitter and a hexadecimal display as the output for the PPM receiver.

Two push button were used as enablers.

All the signals have been forwarded to the logic analyser to watch their state during the time.

The Spark Generator Board

2.1. Study of the Spark Generator

2.1.1 What is a Spark Generator ?

In most vehicle's petrol engine the explosion is provided by a spark. The classic system is to generate it when the piston is at the top of its travel within the cylinder, it is said to be at Top Dead Centre (TDC). It is from here that the angles are measured i.e. it is 0°.

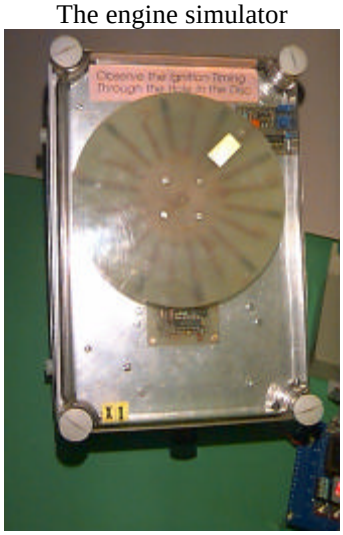
For an optimum explosion, a spark has to occur some time before the Top Dead Centre to allow the mixture of petrol and air to be burning when the piston reach the Top Dead Centre. This is known as the advance angle since the spark occurs «in advance of» Top Dead Centre.

For best engine efficiency, the advance angle is small for low revolution rates and much larger at high revolution rates.

In this project we use a large plastic box containing an electric motor and some electronic components, the whole simulate some aspects of a petrol engine.

With this simulator we set the advance angle at 10 degrees at 3 revolutions per second and about 26 degrees at 10 revolutions per second, with a linear angle between these two rotation rates.

The engine simulator



2.1.2. Tests with the 68HC11

The Digital Systems Laboratory has a complete set of software and hardware package to use the popular Motorola's 68HC11 microcontroller chip.

The software used are :

- Programmer's File Editor (PFE) to enter and edit the C program (source) files
- C68 compiler to translate C to 68HC11 runnable (hex) files
- Download to move the hex file into the 68HC11 memory
- HyperTerminal to communicate with the 68HC11

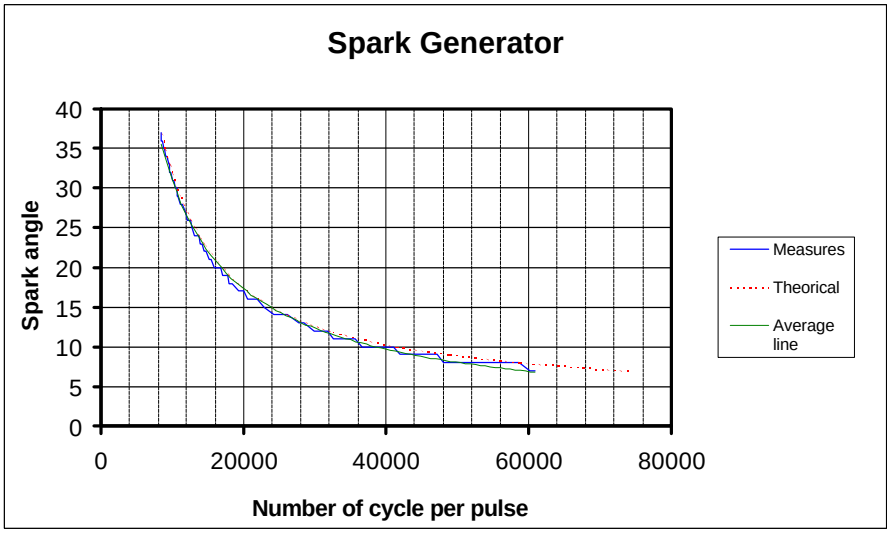
The advantage of the 68HC11 is that we can use the testing board to perform real time test quickly thanks to the C compiler.

In the following C source code, we can see the algorithm used to calculate when the next spark must be generated, as we can see the complexity of the equations used. As far as the floating point format can't be used because it's slow, we have to dispose the equation so that we avoid the integer range limitation.

```
for(;;)
{
   /* Wait for the next sensor pulse */
   while( (PortA & 0x01) == 1);
   while( (PortA & 0x01) == 0);
   t2=TcnT; /* Read the current time */
   if (t2>t1) rotation=t2-t1; /* Calculate the rotation time */
   else rotation=(65535-t1) + t2;
   d=(2*(10000/(rotation/100))+22)/7;
   t_temp=t2+rotation-rotation/360*d;
   Toc1=t_temp;   /* Set at this time. */
   Toc5=t_temp+4; /* Clear at this time. */
   t1=t2;
  }
```

We count the number of clock cycles between each pulse, which calculates the time duration of a single engine revolution. Here a clock cycle is 8us.

Thanks to the HyperTerminal software we can obtain the result of the calculation in real time and use it to create a graphic with Microsoft Excel.

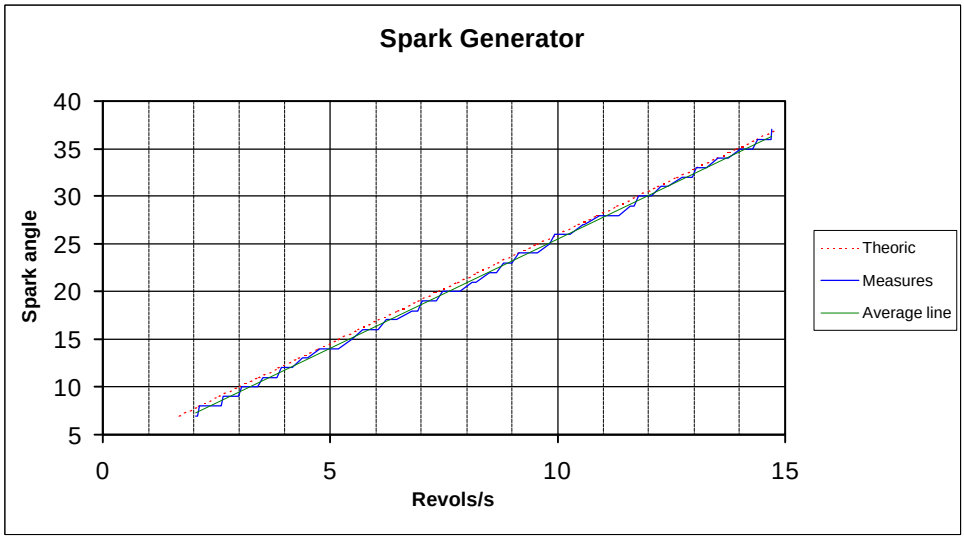


As we can see on the graphic, the measures are very close to the theoretical curve. The result is obtained with the use of two equations including multiplication and divisions on integers.

2.1.3. Implement the spark for VHLD

But the final purpose is to implement the spark algorithm in VHDL and use an Altera chip, which is in fact using a pure logic component. The only division with VHDL is a division by a power of 2, which is in fact the use of a shift register. That means that we can use only linear equations, and so we have to use an other approach to implement the spark angle calculation in VHDL.

Another way is to represent the spark angle in function of the number of revolution per second. In the graphic of this representation, we can see that this representation is linear.

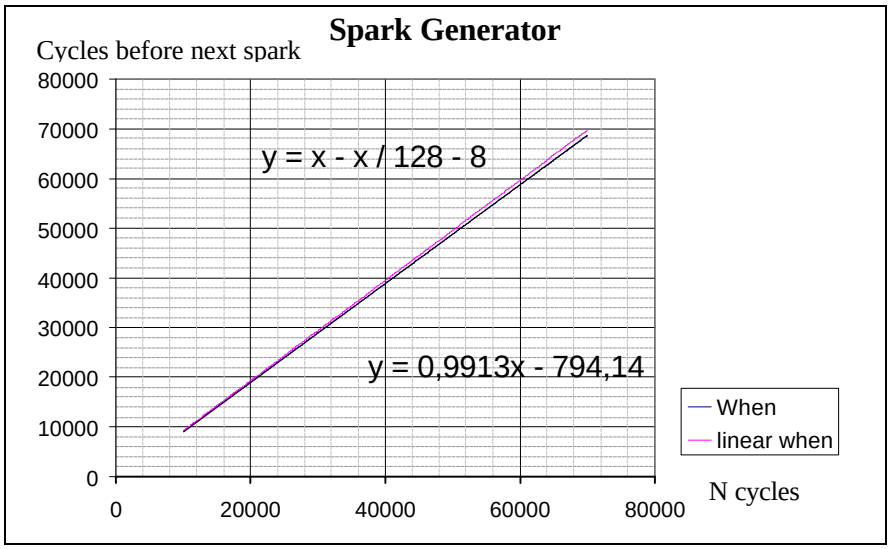


We are looking for the spark angle, but the final information in the algorithm is in fact the time information : «when the spark should occur ?».

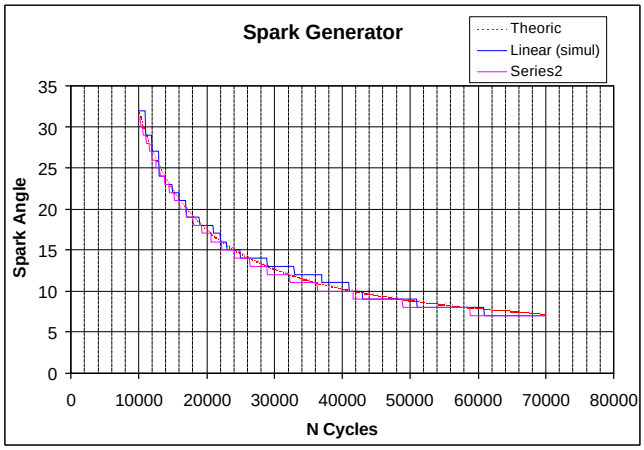
So we represent the time of the spark depending on the duration of a full cycle. The equation of this representation is : y = 0.9913*x – 794,14

As the use of floating variables was too slow with the 68HC11, it is not possible with an Altera chip (whereas it could be possible with larger components).

Hopefully 0.9913 is very close to 1 – 1/128, which can be implemented in logic functions. The resulting equation is : y = x – x / 128 – 8.



The results diverge for longer cycles, which means in fact slower speed, and so it's less critical. The representation spark angle/Number of cycles per pulse show that the angle error never exceeds 1 degree, which is the best result we can attempt from an integer calculation.



The Linear curve represents a first attempt to implement the spark algorithm, which consisted in partitioning the curve in 3 linear parts. But it proved too difficult to implement.

The Series2 curve represents the last choice, more simple and in fact more precise.

2.2. The Spark Generator in VHDL

2.2.1. The components

The basic components used for the VHDL applications are:

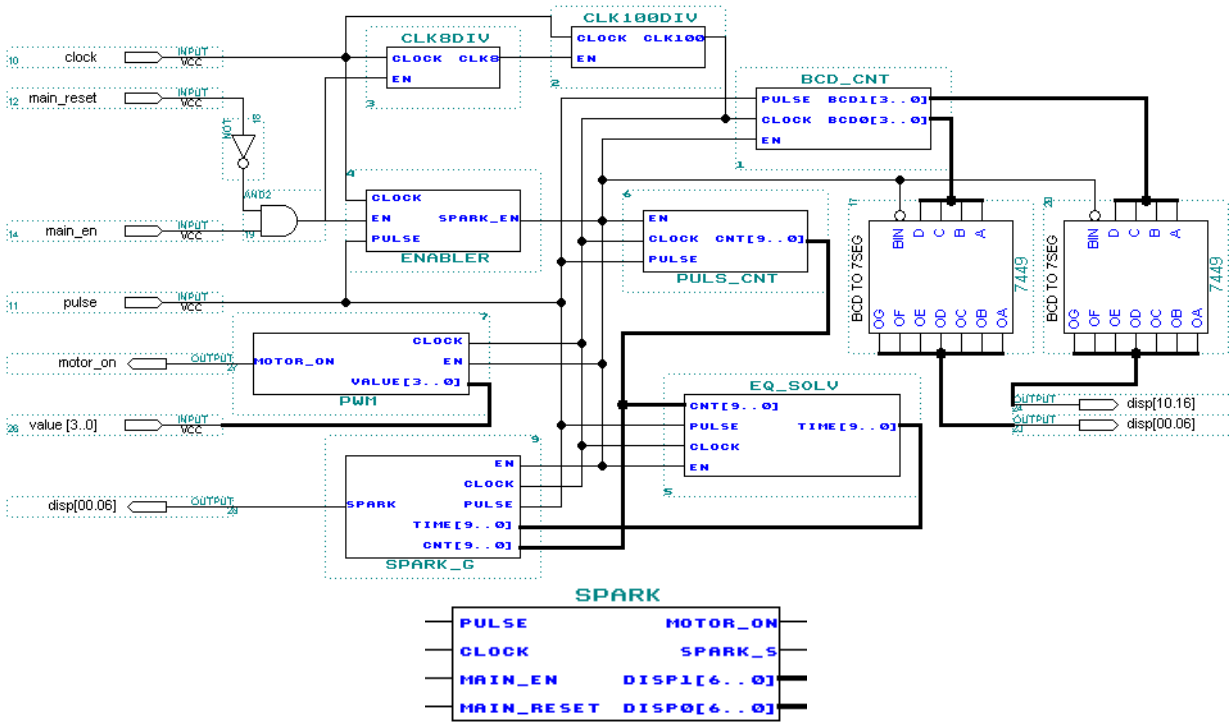
- an enabler, to enable or not the others components
- a pulse counter which counts the number of clock cycles between each pulse
- a solver for the equation, to calculate when the spark must occur
- a spark generator, which generates the spark when necessary
- a PWM generator to modulate the speed of the engine from the board
- a BCD counter, used for the two 7 segment displays and evaluate the speed

The clock used is a 8 MHz clock, so we use 2 clock dividers, to obtain the appropriate clock rate, which is 10 kHz.

2.2.2. The main component

All the VHDL components created are merged in one main file which is represented by this schematic :

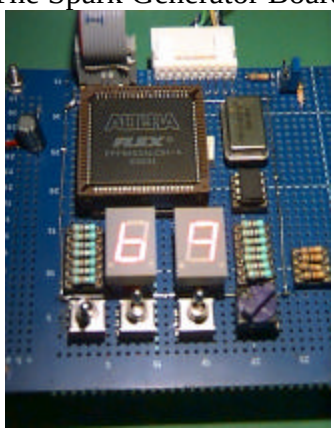
Spark generator : symbolic representation



Finally the component uses 4 input pins and 16 output pins.

2.3. The Spark Generator Board

The Spark Generator Board



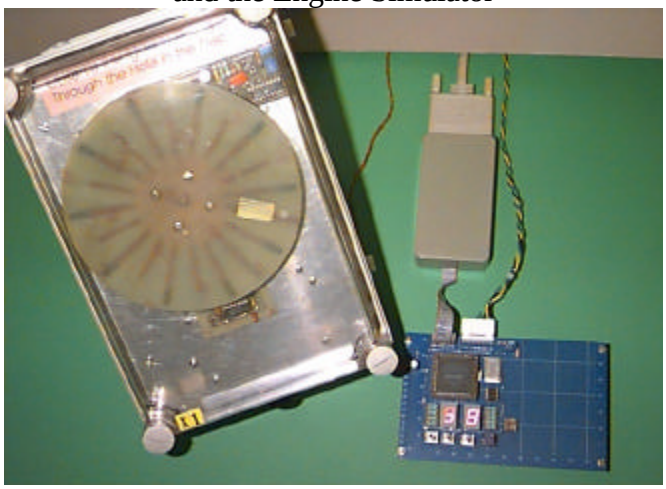
Of course the Altera chip is not efficient alone. So we made a board with various components :

- 3 switches for main_reset, main_en, and speed_en
- a hexadecimal switch, for the PWM generator
- a 8 MHz clock
- a serial EPROM
- a byteblaster socket, to configure the chip
- an input/output socket to connect the engine simulator

The technique used for wiring the board is the wire wrapping method. This is faster to create a prototype board than the classic boards, and the possible future modifications are limited to the physical dimensions of the board.

After a few corrections in the wire wrap connections, and the inversion of the polarity of the displays, the board is working perfectly. A few tests are performed on the board and show that the spark angle correspond to the theoretical value.

The Spark Generator Board
and the Engine Simulator



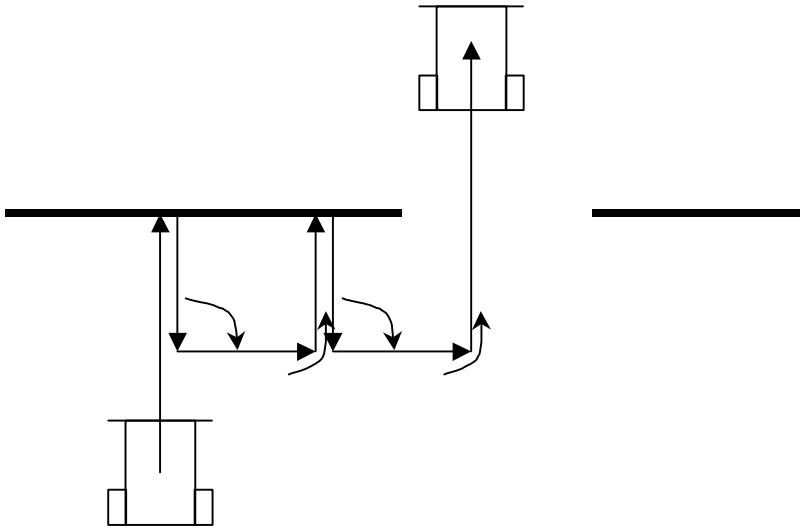
The Path Finder Robot

3.1. Study of the Path Finder Robot

3.1.1. The purpose of the robot

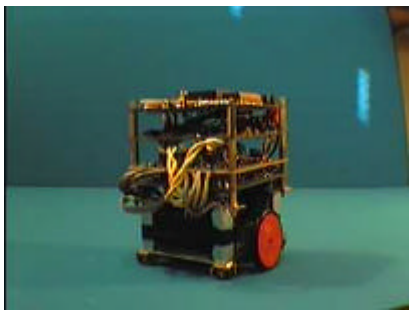
The robot has to find a path across a wall. The robot simply moves forward, if it hits a wall moves backward, turn right, moves forward for short distance, turn left and moves forward again. The robot operates this loop until it find a path, i.e. a hole or a door, and then stops.

The robot finding it's path



The robot is a first approach of more complex robots which can have many applications, the Electronic Engineering Laboratory has built a lots of different Mazemouse robots. The purposes of these robots are to find their way in an unknown labyrinth, there are famous micro-mouse competition in England.

The Kim3 robot built in 1994



More information available at the following URL :
http://www.ukc.ac.uk/electronics

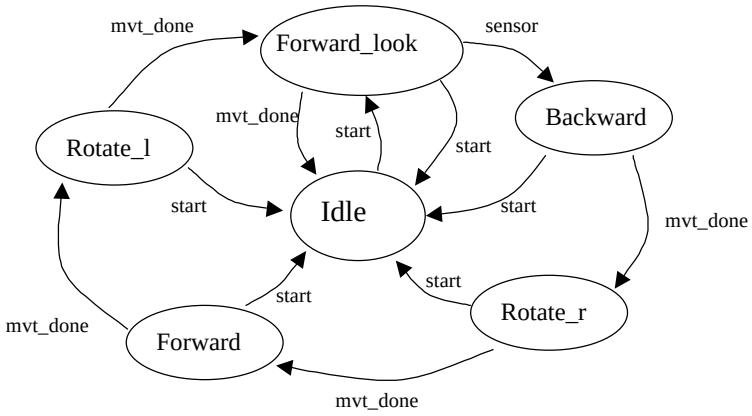
Normally the robots are controlled by a microprocessor, the most current is the 68HC11. This is due to the fact that a robot has to operate complex tasks.

But for this project we will use VHDL, even if it's not the easiest approach, to show that pure logic is able to control a sequential process efficiently.

3.1.2. The Finate State Machine (FSM)

For sequential real-time system we use a Finate State Machine representation which represent the different states of the machine and which variables create the transition from a machine to an other one.

The FSM of the robot



At the beginning the robot is in an Idle state and waits for the start button to be pushed. Then the first state is Forward_look, the robot moves forward until it finds a wall or it's movement is done. If the movement is done, the robot goes back in the Idle state, but if the front sensors detect an obstacle, the robot state machine goes to the next state. The next states create a loop because the robot goes from a state to the next one when the movement is done, this continues until the movement is done in the Forward_look state, then the robot stop.

Each state has a condition on the start push button to stop the robot at any time.

3.1.3. Choice of the hardware

As previously said the robot will be controlled by an Altera device programmed in VHDL.

For the movement we choose to use DC electrical motors, the same as the remote controlled cars, they are big enough to move the robot and carry the batteries.

An important part of the control of the robot is based on it's ability to evaluate the distance reached. The first approach was to use encoded optical wheels linked to the motors, but for mechanical reasons it was difficult to implement, moreover it's not

accurate at all as far as the vibrations will create undesirable pulses and the distance will not be real.

Another idea was to use a PC mouse. The advantages are that it is very accurate and it can avoid the effect of vibrations thanks to the two ways counting technique used, so it can detect if the robot is going either forward or backward.

A PC mouse is normally plugged into the serial port of a computer, in our case we can't use a serial link, we need to take the pulses directly inside the mouse. The pulses were TTL compatible, and so useable with the Altera device.

The mouse works with two power supplies +5V and –12V, so the robot has to provide both.

After a few tests using the 68HC11 board, the mouse reveals to be very accurate and the value obtained for the distance we want to use are around the value 1000, so we need to use at least 12 bits signed counters. In practice we used 16 bit counters, but we could change the size of those counters in the VHDL.

The mouse in place under the robot

3.2. The VHDL part

3.2.1. The components

The basic component used for the VHDL applications are :

- a main component which control the states of the robot
- a controller which control the motors and check if the movement is done

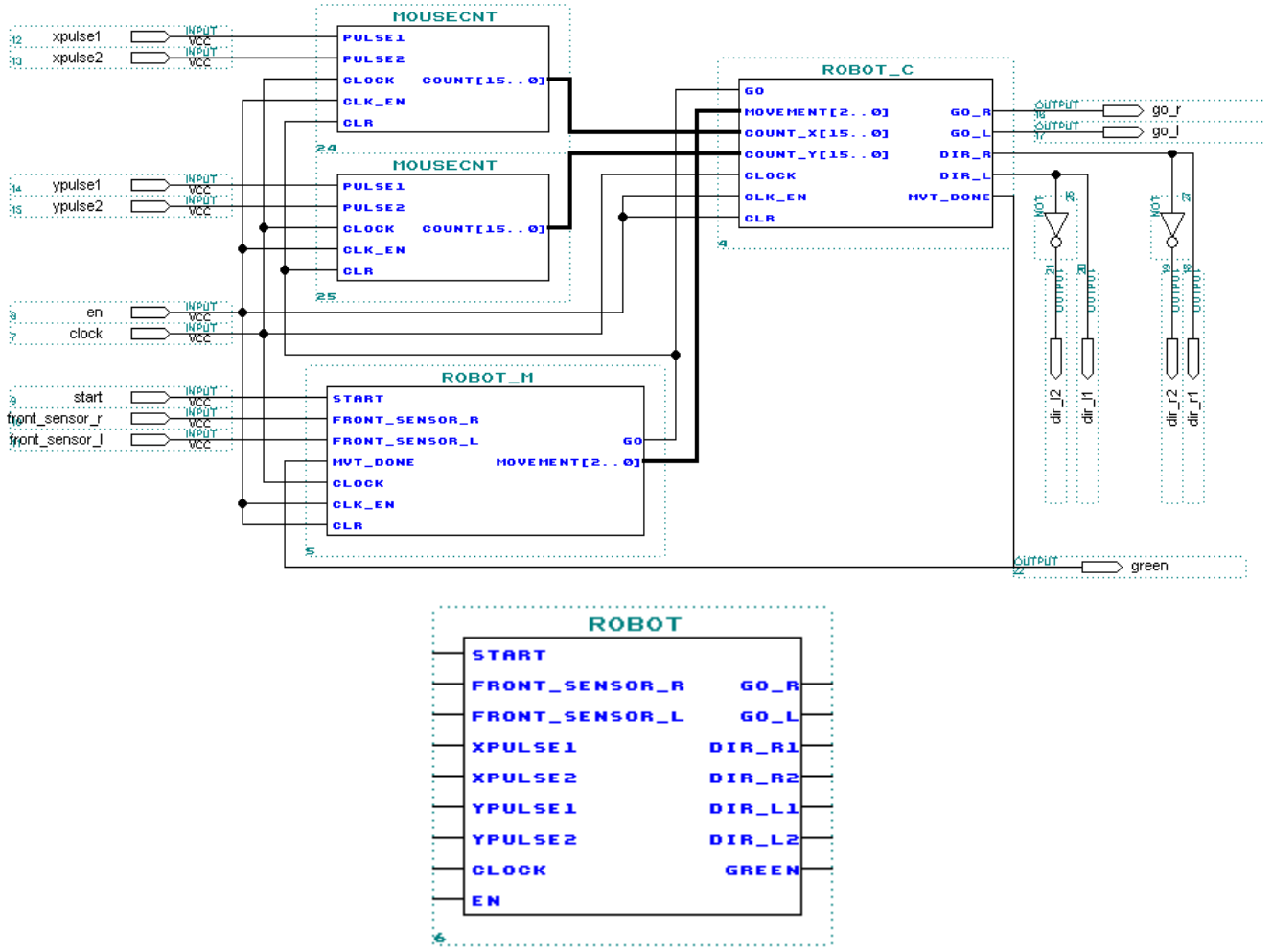
- two mouse counters to count the relative X and Y position of the robot

Here the clock rate is not really important, so we don't have to use any clock divider.

3.2.2. The main component

All the VHDL components created are merged in one main file which is represented by this schematic :

Spark generator : symbolic representation

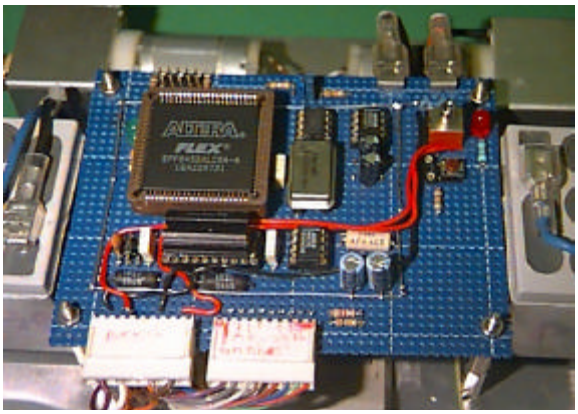


Even if there are less components than with the Spark Generator, there are more inputs and outputs.

There are 9 inputs and 7 outputs.

3.3. The physical part

The Robot board



We cannot drive the motors directly from the Altera chip, so we need various components :

- a switch for the power
- a push button as start
- a 8 MHz clock
- a serial EPROM
- a byteblaster socket, to configure the chip
- two 6V batteries
- a +5V regulator (MAX603)
- a –12V regulator (MAX764) for the mouse
- a bridge driver (L6204) to drive the motors
- an input/output socket to connect the mouse and the sensors
- a socket to connect the motors

The technique used for this board is still the wire wrapping method.

The robot didn’t work immediately, and it appears that the bridge driver needed at least 12V to work.

After adding a second 6V battery and placing the batteries on both sides of the robot to be equilibrate the robot start, but was going only forward and backward and never turned.

After a simulation of the counter, it appears that the counters did not count whereas the VHDL code appeared correct. This error is due in fact to a bug in the Max+PlusII compiler.

Those errors are difficult to anticipate and I spent two days finding an error in the project where the compiler was at fault.

After some modification in the code, the compiler agreed to compile the counters correctly.

Now the robot is doing it’s sequence in the good order : forward, backward, turn right, forward, turn left, forward again until a path.

But those movement are not made correctly : the distances are wrong.

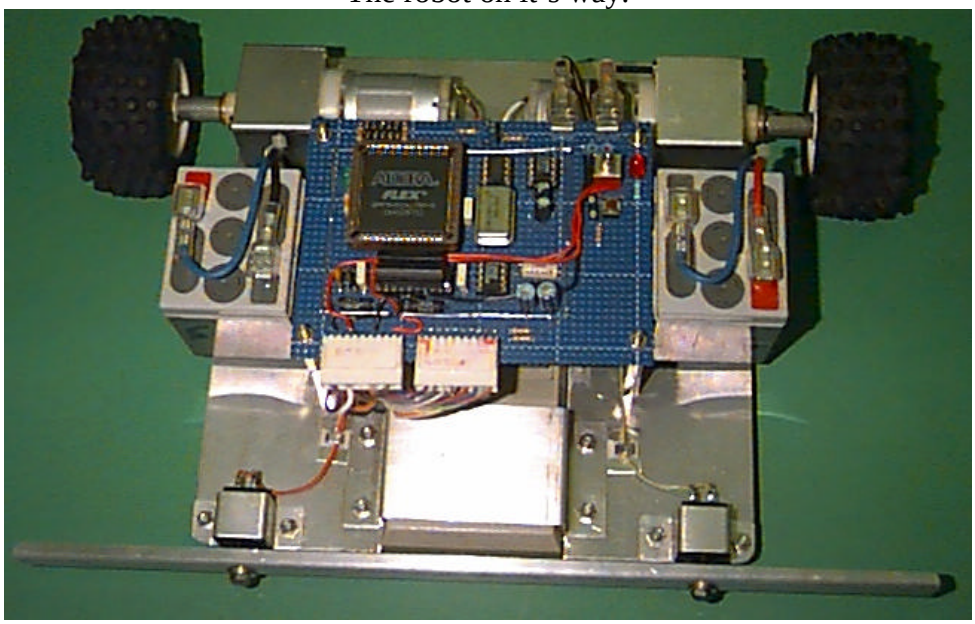
Sometime the robot moves forward for a short distance then and stops, but sometimes does not.

When the robot turns the angle, which should be 90 degrees, but can be less or the robot can turn on itself for a long time (which looks funny).

In fact the counters, which maybe counting correctly now, are disturbed by the noise generated by the electric engines.

A solution would be to separate the engine power from the components power, to reduce the noise.

The robot on it’s way.



CONCLUSION

During this work experiment, I have realised three interesting projects. Even if the robot need some more improvement, all the projects are working correctly.

The use of the VHDL language was an improvement and followed the learning of C and GAL languages, that I learned too use during my two years of DUT.

The first project was a good practice of the design of serial transmission, already studied at the IUT.

The robot and the Spark Generator Board were very interesting for their real time purpose.

I can say that the courses provided at the IUT helped me a lot to accomplish my job.

I have enjoyed the experience of a working experiment in England, and so it could help to find a job more easily during the future, in France or in a foreign country.

Annexes