Reduction of Power Dissipation in FPNI

Hadi JAHANIRAD and Sattar MIRZAKUCHAKI

Iran, College of electrical engineering Iran University of Science and Technology
hadijahanirad@yahoo.com

Iran, College of electrical engineering Iran University of Science and Technology
M_Kuchaki@iust.ac.ir

Abstract: In this paper we present an architecture for nanowire layer in FPNI. The FPNI based on a Field-Programmable Gate Array (FPGA) architecture has two structural part: a CMOS layer and a nano layer which realize routing using nanowires. In this paper we present an approach to build an FPNI chip which is similar to conventional FPGA. By applying genetic algorithm optimization tool, a delay optimized circuit is implemented in FPNI. The power dissipation in chips is compared to a conventional FPGA emphasizing the benefits of this new architecture.

Key words: Field Programmable Gate Array, Field Programmable Nanowire Interconnect, nanoscale, power dissipation.

INTRODUCTION

Using hybrid nano/CMOS architectures have been increased to develop more complex and nanoscale chips. A property of such chips is reduction of power dissipation for an implemented logic circuit on them. By investigating the FPNI architecture we decided to demonstrate its abilities in reducing power consumption.

In this paper we have a quick review on genetic algorithm which is used for optimization purposes in our work in section 1 and then in section 2 we describe the main properties of FPNI architecture. In section 3 we introduce Spartan II FPGA which is used for comparison purposes. We build our FPNI architecture and describe how we implement circuits on it in section 4. In section 5 we introduce our delay and power dissipation calculation methods and finally we present the results of comparing the two chips.

1. Genetic Algorithm

Genetic algorithms (GA) are a group of stochastic optimization techniques. These algorithms work with a set of candidate solutions to the problem (a population of individuals, using GA terminology) and seek to evolve them using concepts derived from genetics and natural selection.

Each individual holds enough information (the genes) to describe a possible solution to the problem. They are evaluated regarding the quality of that solution (i.e. their fitness is computed), and a probabilistic selection method (based on the fitness of each individual) is used to find group of individuals (the parents) that will be used to create the next generation of the population. The individuals of the new generation are created by applying genetic inspired transformations (operators) to the parents.

Among these transformations we can find mutations and crossover. The mutation operator does random changes to the genes, while the crossover combines parts of the genes of two parents to create a single individual.

After multiple iterations, the quality of the population will increase, and when a predetermined stopping condition is met, the solution for the problem will be found on the genes of the best individual of the last generation. There are, of course, multiple variants of this simple framework. The basic GA method is shown in Fig. 1.

Figure 1 Simple Genetic Algorithm

1. Randomly initialize population
2. While stopping condition is not met
   a) Evaluate population
   b) Select parents
   c) Crossover
   d) Mutation
   e) Substitute old population

2. FPNI

Integration of the nanowire crossbars with a CMOS
chip is a common method to develop Micro/nano hybrid architectures (Figure 2) [DEH 02],[ BAR 98].

Each crossbar junction (Figure 1, left) is generally an electrically configurable or reconfigurable device such as an antifuse[3],[DEH 02]. Connection between CMOS gate and nanowires in the crossbar realized via metallic 'pins' on the surface of the chip.

The FPNI (Figure 3), assumes a sea of logic gates, buffers and other components in the CMOS layer, and uses the nanowires only for the interconnection. The large ‘pads’ cover the connection pins, and there is a crossbar rotation so that each nanowire connects to only one pin. Selected junctions act as resistors between nanowires or nanowire-CMOS gate.

Figure 2. Left: crossing nanowires separated by a molecular layer form ‘junctions’ that may be electrically configured as electronic devices. Right: nanowire crossbars connected to a CMOS chip via metallic ‘pins’ on the CMOS surface[DEH 02].

Calculation of dissipated power and delay of an application compiled on FPNI implicated using an electrical model of the nanowires, junctions and CMOS components. For nanowires the model includes the capacitance and resistance per unit length, the closed-junction resistance, and must take into account the geometry of the wires. The CMOS model includes the intrinsic gate delay.

Figure 3 Schematic diagrams of FPNI hybrid circuits

The non-regular nanowire crossbar structure in the FPNI make the estimation of Nanowire capacitance per unit length difficult, moreover the top layer nanowires created by nanoimprint are not parallelepipeds, but undulate as they cross over the spaces between nanowires in the layer beneath (Figure 3). Using Strukov’s model for a regular, parallelepiped nanowire crossbar [STR 05],(where nanowires within a layer are separated by a distance equal to their width), a 3nm thick switching layer separating the two nanowire layers, a nanowire width of 15 nm, and SiO2 as the insulator between and around all nanowires (dielectric constant 3.9), a capacitance per length of approximately 2.8 pF cm−1 will be peridected.

Nanowire pads add additional capacitance, but considering their quite small area compared to that of the nanowires, their contribution is neglected. From these considerations the nanowire capacitance is estimated at 2.0 pF cm−1[SNI 07].

The effective resistivity of the nanowire material is an important agent in determination of Nanowire resistance per unit. Assuming nanowires of copper (the metal specified in the ITRS roadmap), allows us to estimate the resistivity for wires down to 15nm by interpolating the ITRS projections. For example, Cu wires with a line width of 15 nm are interpolated to have an effective resistivity of approximately 8μΩcm, so a square Cu nanowire, 15 nm on a side, would have a resistance of about 355 μΩcm. Nanowire resistivity, ρ, is modeled for very small (<10 nm) wires using[STR 06]:

$$\rho/\rho_0 = 1 + 0.75(1-p)\lambda/d$$  (1)

With p (the fraction of electrons scattered specularly at the surface) assumed to be 0.67, λ (the mean free path) equal to 40nm, ρ0 (the bulk resistivity) equal to 2μΩcm, and d set to the nanowire width. However, this model is known to underestimate the effective resistivity for small wires [STE 02] and assumes negligible increased resistivity due to scattering at grain boundaries (which is possible for very large grain sizes).

An adaptation is made to achieve a more conservative model using Matthiessen’s rule to combine the above surface scattering model with the Mayadas–Shatzkes grain boundary scattering model [STE 02] and assuming an average grain size equal to the nanowire width (which might require annealing to achieve).

Fitting the resulting model to the ITRS resistivity model yield an estimated resistivity of about 24 μΩcm for 4.5nm Cu nanowires; the uncertainty of this estimate, though, is quite high[SNI 07].

Closed-junction resistance depends on the building materials of nanowire crossbar, but experiments illustrated that it is difficult to configure a closed junction to a resistance less than the sum of the
resistances of the nanowires from their junction to their respective drivers. Based on the experimental work [LAU 05] we have chosen a value of 24kΩ as a reasonable estimation of obtainable closed-junction resistance for 30 nm pitch nanowires; [LAU 05]. For higher-resistance, 4.5 nm nanowires, the closed-junction resistance cannot be much less than 120kΩ.

CMOS gate delay was estimated to be 10 ps by noting the projected n-FET switching time of 0.39 ps for the year 2010 from the ITRS roadmap [SNI 04]. Our analysis is not sensitive to this value, though, since circuit timing is strongly dominated by the RC delays of the nanowires.

3. XC2S15

The Spartan-II user-programmable gate array is composed of five major configurable elements [SPA 01]:

- IOBs provide the interface between the package pins and the internal logic
- CLBs provide the functional elements for constructing most logic
- Dedicated block RAM memories of 4096 bits each
- Clock DLLs for clock-distribution delay compensation and clock domain control
- Versatile multi-level interconnects structure.

As can be seen in Figure 4, the CLBs form the central logic structure with easy access to all support and routing structures.

The IOBs are located around all the logic and memory elements for easy and quick routing of signals on and off the chip.

The basic building block of the Spartan-II CLB is the logic cell (LC). An LC includes a 4-input function generator, carry logic, and storage element.

Output from the function generator in each LC drives the CLB output and the D input of the flip-flop. Each Spartan-II CLB contains four LCs, organized in two similar slices; a single slice is shown in Figure (4).

In addition to the four basic LCs, the Spartan-II CLB contains logic that combines function generators to provide functions of five or six inputs. In Figure (5) we can see total CLB arrangement in Spartan II FPGA which would be used in our FPNI architecture.

4. XC2S15_like FPNI

Our purpose for simulation of a XC2S15-like FPNI can be met by definition of CMOS layer of FPNI as a structure with CLB like which exists in XC2S15 FPGA.

Each CL as can be seen in Figure 6 has 15 inputs and 7 outputs which can be accessed by nanowire interconnection layer.
nanowires and 10 vertical nanowires, so for each column in Figure 5 we should have 80(8*10) nanowires and for each row we should have 144(12*12) nanowires. For each row, we should mention 6 IO horizontal nanowires and for each column we need 4 IO nanowires. So for each CLB we have a nanowire structure like Figure 7.

![Figure 7 Nanowire structure for each CLB](image)

We use parameters mentioned in Table 1 for the architecture of our FPNI.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>FPNI 30nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_{nano}</td>
<td>Nanowire pitch</td>
<td>30 nm</td>
</tr>
<tr>
<td>W_{nano}</td>
<td>Nanowire width</td>
<td>15 nm</td>
</tr>
<tr>
<td>R_{closed}</td>
<td>Closed junction resistance</td>
<td>24 kΩ</td>
</tr>
<tr>
<td>σ</td>
<td>Nanowire resistivity</td>
<td>8 μΩ/μm</td>
</tr>
</tbody>
</table>

The presented architecture for Spartan II like FPNI has very simple structure. For example we ignored switch boxes and connect directly all pins to a dedicated nanowire and for each horizontal or vertical nanowire we defined a nano switch in each cross-section with vertical or horizontal nanowire. Because of this simple architecture we deal with some delay and routing problem that will not be optimized. We can now compare delay and power dissipation for a given circuit in this chip and Spartan II FPGA.

5. Case study

As we have such a FPNI chip we can implement variety of logic circuits on it. For our work we select 4-bit alu (74181) to be implemented on this chip. Logic diagram for this IC is presented in Figure 14. For implementation of this circuit on our FPNI CLBs we partitioned this circuit to several segments each can be implemented on single CL, so we have 11 CL in the best case.

Next stage in our work is creating a delay optimized implementation of this adapted circuit on FPNI chip.

We started our work by creating a placement file for this logic circuit (Figure 8). This file contains several columns:

First column is a 12 bit number which determines necessary properties of one of sources. The MSB indicates if the source is an IO or a CL output source. Bits 6 to 12 indicate the number of CL where the source is originated. Bit number 5 indicates that the nanowire relates to this pin is horizontal or vertical.

<table>
<thead>
<tr>
<th>Source ID</th>
<th>Bit 0</th>
<th>Bit 1-4</th>
<th>Bit 5</th>
<th>Bit 6-12</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO</td>
<td>0</td>
<td>Pin CL</td>
<td>H/V</td>
<td>N CL</td>
</tr>
<tr>
<td>CL Output</td>
<td>1</td>
<td>Pin CL</td>
<td>H/V</td>
<td>N CL</td>
</tr>
</tbody>
</table>

Pin numbers assign to CL pins as be shown in Figure 9.

![Figure 8 Placement file for alu4(74181)](image)

Bits 1 to 4 is the pin of CL related to that IO source. Other columns contain 9 bit numbers which determine the properties of sinks. Bits 6 to 9 indicate the number of CL related to that sink. Bit number 5 indicates that related nanowire is horizontal or vertical. Bits 1 to 4 is the pin of CL related to that sink. Pin numbers assign to CL pins as be shown in Figure 9.

Next stage is assignment of the chip CL to 11 CL mentioned in the placement file. The goal of our work is optimum assignment to have minimum delay in our
implementation. Our optimization method is based on GA, so we define an initial population which determines the required CL numbers, for example:

\[
\text{pre}_\text{PlaceMat} = [106 \ 127 \ 128 \ 129 \ 130 \ 131 \ 132 \ 133 \ 134 \ 135 \ 136]
\]

IO positions was determined before and presented in a matrix. Considering this information a matrix would be created which determines the nanowire for each source and sink, this matrix named NW_related matrix.

IO positions was determined before and presented in a matrix. Considering this information a matrix would be created which determines the nanowire for each source and sink, this matrix named NW_related matrix.

Each source is connected to sinks illustrated in the same row of source in placement file. So we have four types connection:

1) Source is horizontal and sink is vertical Figure (10).
2) Source is horizontal and sink is horizontal Figure (11).
3) Source is vertical and sink is horizontal Figure (12).
4) Source is vertical and sink is vertical Figure (13).

For case (1) we should determine the position of one switch (SWso):

\[
\text{SWso}_\text{pos}_x = x_{\text{sink}}, \text{SWso}_\text{pos}_y = y_{\text{source}} \quad (1)
\]

For case (2) we should select an unused vertical nanowire to connect two horizontal nanowires. So we have two switches (SWso,SWsi) which have position determined by following equations:

\[
\text{SWsi}_\text{pos}_x = \text{VerNW}_x, \text{SWsi}_\text{pos}_y = y_{\text{sink}} \quad (2)
\]

\[
\text{SWso}_\text{pos}_x = \text{VerNW}_x, \text{SWso}_\text{pos}_y = y_{\text{source}} \quad (3)
\]

For case (3) we should determine the position of one switch (SWso):

\[
\text{SWso}_\text{pos}_x = x_{\text{source}}, \text{SWso}_\text{pos}_y = y_{\text{sink}} \quad (4)
\]

For case (4) we should select an unused horizontal nanowire to connect two vertical nanowires. So we have two switches (SWso,SWsi) which have position determined by following equations:

\[
\text{SWsi}_\text{pos}_x = x_{\text{sink}}, \text{SWsi}_\text{pos}_y = \text{HorNW}_y \quad (5)
\]

\[
\text{SWso}_\text{pos}_x = x_{\text{source}}, \text{SWsi}_\text{pos}_y = \text{HorNW}_y \quad (6)
\]
Next stage is delay calculation. For this purpose we use Elmore delay model to estimate delay for each path through nanowires and junctions (see Figure 15) [ELM 48][CON 96][OKA 96]. Since we have four types of connection between sources and sinks, for each source we first determine the related sinks from routing file and then we determine the type of connection between the source and each sink. We partitioned delay calculation for each source in two stages: first we calculated delay from each sink to SWso and write it in a matrix then we calculate delay from each SWso to source and write it in another matrix. Final delay matrix for each source_sink connection can be made by addition of two last matrices. In optimization we use this resulted matrix for determination of scores in the GA and positions of required CLBs as the population. We run gatool of MATLAB for optimization with Table 2 options.

Result for positions of required CLBs is as:
CL_Position = [99 10 9 23 124 4 26 91 49 56 8]

<table>
<thead>
<tr>
<th>Fitness Scaling</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Roulette</td>
</tr>
<tr>
<td>Mutation</td>
<td>Gaussian</td>
</tr>
<tr>
<td>Crossover</td>
<td>Scattered</td>
</tr>
<tr>
<td>Migration</td>
<td>Forward(0.2)</td>
</tr>
<tr>
<td>Initial population</td>
<td>20</td>
</tr>
</tbody>
</table>

Dynamic power analysis tallies the number of nanowires allocated by our simulation to implement the circuit onto an FPNI target, and computes the dynamic power required to charge and discharge the capacitance of those wires using the formula [SNI 07]

\[
\text{Dynamic Power} = \frac{1}{2} A N C V_{dd}^2 f
\]

Where A is the average ‘activity’ of a signal, N is the number of allocated nanowires, C is the capacitance of a single nanowire, V_{dd} is the supply voltage used by the CMOS, and f is the maximum clock frequency determined by timing analysis. We have chosen an activity of 0.1, following Davis [DAV 98] and V_{dd} of 1.0 V from the ITRS roadmap for the year 2010. Note that we have not computed the static power dissipation in the CMOS gates and neither flipflops nor the dynamic power needed to drive the CMOS clock tree.

In the next stage we want to compare our results with XILINX synthesis and simulation software (ISE 8.2). So in a schematic based project we created a logic circuit like that shown in Figure 14.

After synthesizing this circuit, the software produced detailed delay information for all nodes in placed and routed circuit in XC2S15 FPGA. We present 20 worst node delays obtained from Xilinx software and from our own FPNI chip in Table 3. As can be seen, there is a similarity between delays in two chips. In Table 4 we compare power dissipation.

We see that we have a great reduction in power dissipation for this chip. For calculating this power dissipation we consider the path delay for our FPNI chip (we show 5 critical path delays in Table 5 and resulted from this table we obtain that:

\[
f = \frac{1}{3 \times 561 \text{ ns}} = 280 \text{ MHz}
\]

6. Conclusions

In this paper we have demonstrated a simple chip derived from the FPNI idea presented by HP. We construct a FPNI chip which has two parts: a CMOS layer (CMOS FPGA selected was Spartan II), and a crossbar nanowire layer which is used for routing.

By considering this architecture we implemented an instance logic circuit in Spartan II FPGA and Spartan II_like FPNI. We conclude from our compilation and ISE compilation that:

In two cases we have approximately similar delays, so we have similar applied frequency in two cases. But regarding the power dissipation we can see a dramatic reduction in FPNI. This can be a great improvement that by this rude architecture for routing system we can achieve same applied frequency and reduced power dissipation. In future work we will enhance nanowire routing architecture to reduce delay and area.
Table 3  The 20 worst nod delays for chips

<table>
<thead>
<tr>
<th>Nod number</th>
<th>XC2S15(ns)</th>
<th>FPNI(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.183</td>
<td>3.4190</td>
</tr>
<tr>
<td>2</td>
<td>1.977</td>
<td>3.3610</td>
</tr>
<tr>
<td>3</td>
<td>1.960</td>
<td>3.2320</td>
</tr>
<tr>
<td>4</td>
<td>1.860</td>
<td>3.0860</td>
</tr>
<tr>
<td>5</td>
<td>1.852</td>
<td>2.5480</td>
</tr>
<tr>
<td>6</td>
<td>1.753</td>
<td>2.4020</td>
</tr>
<tr>
<td>7</td>
<td>1.750</td>
<td>1.8490</td>
</tr>
<tr>
<td>8</td>
<td>1.697</td>
<td>1.5560</td>
</tr>
<tr>
<td>9</td>
<td>1.635</td>
<td>1.4790</td>
</tr>
<tr>
<td>10</td>
<td>1.606</td>
<td>1.3220</td>
</tr>
<tr>
<td>11</td>
<td>1.547</td>
<td>1.2780</td>
</tr>
<tr>
<td>12</td>
<td>1.531</td>
<td>1.2680</td>
</tr>
<tr>
<td>13</td>
<td>1.459</td>
<td>1.2500</td>
</tr>
<tr>
<td>14</td>
<td>1.428</td>
<td>1.2490</td>
</tr>
<tr>
<td>15</td>
<td>1.401</td>
<td>1.2280</td>
</tr>
<tr>
<td>16</td>
<td>1.320</td>
<td>1.2270</td>
</tr>
<tr>
<td>17</td>
<td>1.273</td>
<td>1.1820</td>
</tr>
<tr>
<td>18</td>
<td>1.128</td>
<td>1.1730</td>
</tr>
<tr>
<td>19</td>
<td>1.035</td>
<td>1.1390</td>
</tr>
<tr>
<td>20</td>
<td>1.034</td>
<td>1.1050</td>
</tr>
</tbody>
</table>

Table 4  Critical Path Delays for FPNI

<table>
<thead>
<tr>
<th>source</th>
<th>sink</th>
<th>Path delay(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>Cn+4</td>
<td>3.01</td>
</tr>
<tr>
<td>A0</td>
<td>A=B</td>
<td>2.902</td>
</tr>
<tr>
<td>B1</td>
<td>Cn+4</td>
<td>2.93</td>
</tr>
<tr>
<td>A2</td>
<td>Cn+4</td>
<td>3.561</td>
</tr>
<tr>
<td>M</td>
<td>F3</td>
<td>3.283</td>
</tr>
<tr>
<td>Cin</td>
<td>A=B</td>
<td>3.05</td>
</tr>
</tbody>
</table>

Table 5  Power dissipation Comparison

<table>
<thead>
<tr>
<th></th>
<th>XC2S15</th>
<th>FPNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Diss(mW)</td>
<td>9</td>
<td>0.54</td>
</tr>
</tbody>
</table>

7. References:


[STR 06] D B. Strukov and K K. Likharev,”A reconfigurable architecture for hybrid CMOS/nanodevice circuits”, FPGA ’06 (Monterey, CA, USA, Feb. 2006)
Figure 14 ALU4 Logic diagram

Figure 15 Electrical model of nanowires and junctions. A signal with a fanout of 2 (a) is implemented by electrically closing junction switches between the nanowire driven by the source and the two nanowires connected to the sinks (b). The electrical model used for estimating signal delay (c) uses the physical coordinates of the nanowires and their closed junctions to derive nanowire resistance and capacitance. Delay is estimated using the Elmore delay model; in this example, delay (source→sink1)
\[ \tau = R_d (4D + d_1/2 + d_2 + d_3)C + 2\pi d \omega C + R_d (D + d_4/2)C \]