BAB II
TINJAUAN PUSTAKA

Bagian utama yang akan dibahas dalam sistem FPU adalah bilangan *floating point* dalam pendekatan standar IEEE 754. Selanjutnya yang akan dibahas adalah aritmatika bilangan *floating point*, dan *hardware* yang akan digunakan dalam implementasi FPU.

2.1 Bilangan *Floating Point*

Bilangan *floating point* adalah sebuah bilangan yang dapat digunakan untuk menggambarkan sebuah nilai yang sangat besar atau sangat kecil (bilangan pecahan desimal atau bilangan yang memiliki koma dan pangkat). Bilangan ini dapat direpresentasikan menjadi dua bagian, yakni bagian ***mantisa*** dan bagian ***eksponen***. Bagian mantisa menentukan digit dalam angka tersebut, sedangkan eksponen menentukan nilai berapa besar pangkat pada bagian mantisa tersebut (jarak dari posisi titik desimal). Untuk memudahkan pengertiannya, contoh:

1) Misalkan terdapat sebuah bilangan 7052000000 maka bilangan ini dapat dituliskan dalam bentuk bilangan *floating point*: $7052\text{E}6$ yang secara matematis artinya: 7052×10^6 Bagian mantisanya adalah 7052 bagian eksponennya adalah $\text{E}6$
2) Misalkan terdapat sebuah bilangan 0.00000944 maka bilangan ini dapat dituliskan dalam bentuk bilangan floating point: $944\text{E-}5$ yang secara matematis artinya: 944×10^{-5} Bagian mantisanya adalah 944 bagian eksponennya adalah $\text{E-}5$.

Bilangan *floating point* tidak selalu dapat memberikan hasil eksak, karena dapat terjadi pembulatan. Bilangan *floating point* digunakan dalam perhitungan bilangan yang sangat besar dan atau bilangan yang sangat kecil.

2.2 *Floating Point* Standar IEEE 754

Bentuk standar *floating point* IEEE 754 awalnya ditentukan pada tahun 1985, kemudian mengalami revisi pada tahun 2008. Standar IEEE 754 dijadikan standar aritmatika bilangan *floating point* yang didukung oleh seluruh CPU.

Ide pembentukan standar IEEE 754 adalah dari bentuk representasi bilangan pecahan biner dalam bentuk *floating point* $V = x \times 2^y$, bentuk tersebut ingin direpresentasikan dengan hanya memberikan nilai x dan y-nya saja. Representasi dari bilangan *floating point* standar IEEE 754 adalah:

$V = (-1)^s \times M \times 2^E$ ..... (1)

Dengan:

- Bit tanda S menentukan bilangan tersebut positif atau negative.
- Mantisa M adalah bilangan pecahan, berkisar antara 1 dan $2 - \varepsilon$ atau antara 0 dan $1 - \varepsilon$
- Eksponen E adalah bobot nilai bilangan.

Kode biner bilangan *floating point* dalam format IEEE 754 terdiri dari 3 bidang, yaitu: bit tanda, eksponen, dan pecahan tanpa bit paling signifikan (*most significant bit*). Gambar 2.1 menunjukkan tiga bidang float dalam format IEEE 754.

Gambar 2.1. Tiga bidang float dalam format IEEE 754

Sumber: http://en.wikipedia.org/wiki/IEEE_754-1985

Eksponen tidak disimpan dalam bentuk komplemen dua, tetapi dalam bentuk format bias (*offset biner*).

$E = e - bias$ ..... (2)

e adalah nilai tak bertanda (*unsigned*) yang dikonversikan langsung dari eksponen, dan $bias = 2^{n-1} - 1$ dimana n adalah banyaknya bit eksponensial. Misalnya panjang bit eksponen sial adalah 8 bit maka $bias = 2^{8-1} - 1 = 127$, $e = 0$ sampai 254, $E = -126$ sampai 127.

MSB (*most significant bit*) dalam signifikan tidak disimpan tetapi dapat ditentukan dari nilai eksponen. Jika $0 < eksponen < 2^{E-1}$, maka MSB dari signifikan adalah 1, dan nilai ini dinamakan ternormalisasi. Jika eksponen 0 dan signifikan tidak 0 maka MSB dari signifikan adalah 0 maka nilai ini dikatakan ter-denormalisasi. Dan tiga kasus khusus lainnya yang muncul adalah:

1) Jika eksponen adalah 0 dan mantisa (*fraction)* adalah 0, maka nilainya adalah ± 0 (tergantung dari bit tanda).

2) Jika eksponen$= 2^{e-1}$ dan mantisa adalah 0, maka nilainya adalah $\pm\infty$ (tergantung dari bit tanda).
3) Jika eksponen$= 2^{e-1}$ dan mantisa tidak sama dengan 0, maka nilai yang diwakilkan adalah bukan angka (*Not a Number*).

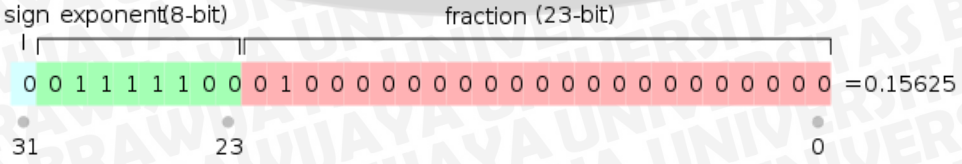
Pembulatan pada bilangan *floating point* terdapat 4 mode yaitu:

1) Pembulatan terdekat, adalah pembulatan ke nilai yang terdekat, jika nilai tersebut berada ditengah-tengah maka dibulatkan dengan mengacu nilai LSB agar menjadi genap (0).
2) Pembulatan 0, adalah pembulatan yang diarahkan ke nilai 0 atau disebut juga dengan pemotongan.
3) Pembulatan $+\infty$, adalah pembulatan yang diarahkan menuju nilai tak terhingga positif.
4) Pembulatan $-\infty$, adalah pembulatan yang diarahkan menuju nilai tak terhingga negatif.

2.3 *Floating Point* Presisi Tunggal

Dalam sistem bilangan *floating point* standar IEEE 754, format bilangan dibagi menjadi bilangan *floating point* biner dan desimal. Format bilangan *floating point* biner dibagi menjadi empat yaitu, biner 16 (*half precission*), biner 32 (*single precission*), biner 64 (*double precission*), dan biner 128 (*quadruple precission*). Bilangan *floating point* biner 32 atau presisi tunggal disimpan dalam 32 bit, yang terdiri dari 1 bit *sign*, 8 bit eksponen, dan 23 bit mantissa.

Mantissa pada format *floating point* standar IEEE 754 memiliki bit tersembunyi yang bernilai 1 kecuali jika semua bit eksponent bernilai nol. Bit tersembunyi ini menjadi MSB pada nilai mantisa. Dengan 23 bit mantisa yang muncul pada format memori, maka total presisi untuk mantisa adalah 24 bit atau sekitar 7 digit desimal ($7 \log_{10} 2 \approx 7.224$). Susunan bit bilangan *floating point* presisi tunggal ditunjukkan dalam Gambar 2.2.



Gambar 2.2. Susunan bit bilangan *floating point* presisi tunggal

Sumber: http://en.wikipedia.org/wiki/IEEE_754-1985.htm

Nilai riil yang diasumsikan oleh data yang diberikan bilangan *floating point* presisi tunggal dengan memperhitungkan eksponen bias dan bit tersembunyi mantissa adalah $= (-1)^{sign}(1.b_{-1}b_{-2} \dots b_{-23})_2 \times 2^{e-127}$ atau lebih tepatnya adalah:

$$V = (-1)^{sign}\left(1 + \sum_{i=1}^{23}[b_{-i}]\,2^{-i}\right) \times 2^{(e-127)} \quad \text{..... (3)}$$

Dengan:

- $sign$ adalah bit tanda
- e adalah eksponen sebesar 8 bit tersimpan dalam bidang eksponen.
- b adalah mantissa sebesar 23 bit terletak pada bidang mantissa.

Eksponen bilangan *floating point* presisi tunggal dikodekan dengan bias eksponen dalam standar IEEE 754, besar bias tersebut adalah 127. Contoh representasi tersebut adalah:

- $E_{min}(1) = -126$
- $E\,(50) = -77$
- $E\,(127) = 0$
- $E_{maks}(254) = 127$

Jadi seperti yang didefinisikan oleh representasi bias eksponen, dalam rangka untuk mendapatkan nilai eksponen riil maka eksponen tertulis harus dikurangkan dengan bias eksponen yaitu bernilai 127.

Catatan untuk bilangan *floating point* presisi tunggal adalah sebagai berikut:

1) Perhitungan nilai angka denormalisasi sama dengan angka ternormalisasi kecuali bahwa $e = -126$ dan $m_{53} = 0$ ($e \neq -127$ karena mantisa harus lebih bergeser kekanan sebanyak 1 bit, dalam rangka untuk memasukkan bit paling signifikan (MSB), yang tidak selalu satu dalam hal ini. Hal ini diimbangi oleh penambahan satu ke eksponen menjadi -126 untuk perhitungan).
2) -126 adalah ekponen terkecil untuk nilai ternormalisasi
3) Terdapat dua Nol, yaitu $+0$ dan -0 (tergantung nilai bit tanda). Nol positif tertulis 0000 0000h sedangkan nol negatif tertulis 8000 0000h.

4) Terdapat dua nilai tak terhingga, yaitu $+\infty$ dan $-\infty$ (tergantung bit tanda).

   $7100\ 0000h = \infty$

   $F100\ 0000h = -\infty$

5) NaN (*Not a Number*) mungkin mempunyai tanda (positif atau negatif) dan mantisa, tapi ini tidak memiliki arti selain untuk diagnosa kesalahan. NaN direpresentasikan ketika semua bit eksponen bernilai satu dan nilai mantissa tidak sama dengan nol.

6) Nilai positif dan negatif paling dekat dengan nol untuk nilai yang terdenormalisasi (diwakili dengan semua nilai 0 di bidang ekponen dan nilai biner 1 dalam bidang mantisa) adalah:

   $0000\ 0001h = 2^{-126-23}$

   $\approx 1.4 \times 10^{-45}$

   $8000\ 0001h = -2^{-126-23}$

   $\approx -1.4 \times 10^{-45}$

7) Nilai positif dan negatif terbesar yang direpresentasikan oleh bilangan *floating point* terdenormalisasi adalah:

   $007F\ FFFFh = (1 - 2^{-23}) \times 2^{-126}$

   $\approx 1{,}1754942 \times 10^{-38}$

   $807F\ FFFFh = -(1 - 2^{-23}) \times 2^{-126}$

   $\approx -1{,}1754942 \times 10^{-38}$

8) Nilai positif dan negatif paling dekat dengan nol untuk nilai yang ternormalisasi (diwakili dengan nilai 1 di bidang ekponen dan nilai 0 di bidang mantisa) adalah:

   $0100\ 0000h = 2^{-126} \approx 1{,}1754943 \times 10^{-38}$

   $8100\ 0000h = -2^{-126} \approx -1{,}1754943 \times 10^{-308}$

9) Nilai positif dan negatif yang terjauh dari nol (diwakili dengan nilai 254 di bidang eksponen dan semua bit bernilai 1 di bidang mantisa) adalah:

   $7F7F\ FFFFh = (1 + (1 - 2^{-23})) \times 2^{127}$

   $\approx 3{,}4028234 \times 10^{38}$

   $FF7F\ FFFFh = -(1 + (1 - 2^{-52})) \times 2^{23}$

   $\approx -3{,}4028234 \times 10^{38}$

2.4 Aritmatika Bilangan *Floating Point*

Pada dasarnya aritmatika bilangan *floating point* sama dengan aritmatika bilangan ilmiah pada umumnya. Pada penjumlahan dan pengurangan prosesnya lebih rumit daripada pada perkalian dan pembagian, karena jika eksponen bilangan yang akan dijumlah berbeda maka eksponennya harus disetarakan terlebih dahulu, atau bilangan dengan eksponen lebih kecil harus disamakan dengan eksponen bilangan lain. Perhitungan dua bilangan A dan bilangan B dengan hasil bilangan R yang masing-masing terdiri dari bit tanda (S_A, S_B, dan S_R), eksponen (E_A, E_B, dan E_R), dan pecahan (M_A, M_B, dan M_R).

Langkah-langkah utama dalam perhitungan R jumlah atau selisih dua bilangan *floating point* A dan B adalah sebagai berikut:

1) Menghitung nilai absolut dari selisih kedua eksponen, yaitu $|E_A - E_B|$, dan mengatur E_R eksponen hasil dari eksponen *operand* yang nilainya lebih besar.
2) Lakukan pergeseran signifikan dengan eksponen yang lebih kecil ke kanan sebanyak $|E_A - E_B|$.
3) Tambahkan atau kurangkan M_A dan M_B sesuai dengan operasi yang efektif berdasarkan operasi yang dinginkan dan kedua tanda negatif *operand* (S_A dan S_B), dan dapatkan mantisa hasil perhitungan M_R sekaligus tanda negatif hasil perhitungan S_R.
4) Normalisasi M_R dan sesuaikan nilai E_R-nya, dan bulatkan nilai M_R yang mungkin memerlukan pengkajian ulang terhadap nilai E_R.

Prosedur ini cukup umum dan dapat mengalami berbagai modifikasi. Sedangkan langkah-langkah utama dalam perhitungan R hasil perkalian dari perkalian dua bilangan A dan B adalah sebagai berikut:

1) Menghitung eksponen dari hasil perhitungan $E_R = E_A + E_B - E_{BIAS}$.
2) Kalikan M_A dan M_B untuk mendapatkan hasil dari mantisa M_R.
3) Normalisasi M_R dan sesuaian E_R. Bulatkan nilai M_R yang mungkin memerlukan pengkajian ulang terhadap nilai E_R.

Setelah penjumlahan dan pengurangan, perkalian merupakan operasi yang paling sering dilakukan dalam komputasi ilmiah. Dan secara umum pembagian jauh lebih jarang terjadi dibandingkan dengan operasi-operasi sebelumnya.

Langkah-langkah penting dalam perhitungan R hasil dari bilangan A dibagi bilangan B adalah sebagai berikut:

1) Menghitung eksponen dari hasil perhitungan $E_R = E_A - E_B + E_{BIAS}$.
2) Bagi nilai M_A dengan M_B untuk mendapatkan hasil dari mantisa M_R.
3) Normalisasi M_R dan sesuaian E_R. Bulatkan nilai M_R yang mungkin memerlukan pengkajian ulang tehadap nilai E_R.

Dalam implementasinya pada rangkaian digital, akan terjadi kesalahan perhitungan jika salah satu atau kedua bilangan bernilai khusus, misalnya 0. Hal ini terjadi tidak hanya dalam perhitungan bilangan *floating point* saja namun dalam bilangan bulat atau *fixed point* juga bisa terjadi.

Karena hal itu maka dalam sistem aritmatika *floating point* terdapat unit penangan khusus untuk masukan bernilai tertentu. Unit penangan khusus ini biasa disebut dengan *exception handler*.

IEEE standar mendefinisikan 5 jenis pengecualian yang harus diisyaratkan melalui *flag status* ketika ditemui operasi yang membutuhkan penanganan khusus. Antara lain:

2.4.1 *Invalid Operation*

Beberapa operasi aritmatika adalah tidak valid, misalnya adalah 0 dibagi 0. Hasil perhitungan ini akan menghasilkan keluaran berupa *Not a Number*. Ada 2 jenis *Not a Number* (NaN) yang didefinisikan menurut standar IEEE 754 yaitu *Quiet-NaN* dan *Signaled-NaN*. Keduanya memiliki format data berikut:

$$QNaN = s\ 11111111\ 11111111111111111111111$$
$$SNaN = s\ 11111111\ xxxxxxxxxxxxxxxxxxxxxxx$$

Dimana s adalah tanda negatif, dan deret x pada SNaN adalah 0 atau 1 dengan catatan, minimal satu x yang bernilai 1.

2.4.2 *Division By Zero*

Dalam matematika, pembagian ini disebut pembagian dengan nol ketika nilai pembagi adalah 0. Pembagian ini dapat dinyatakan sebagai $a\ /\ 0$, dimana a adalah bilangan terbagi (dividen). Apakah persamaan ini dapat diberi nilai keluaran yang terdefinisi dengan baik (tak berhingga) tergantung pada pengaturan sistem aritmatika.

Dalam penulisan biasa (bilangan real) aritmatika, ekspresi tidak memiliki makna. Tetapi dalam pemrograman computer, pembagian integer dengan nol dapat menyebabkan sebuah program berhenti, atau dalam kasus pembagian dengan nol bilangan *floating point*, dapat menghasilkan nilai bukan angka khusus (dalam hal ini adalah tak berhingga).

Pembagian angka bukan nol dengan nol akan memberikan keluaran berupa tak berhingga (*infinity*). Penambahan atau perkalian dua angka tertentu bisa juga menghasilkan *infinity*. Jadi untuk membedakannya maka diberikan sinyal keluaran khusus yang dinamakan *division by zero*.

2.4.3 *Inexact*

Inexact adalah jenis pengecualian yang mengisyaratkan ketika hasil operasi aritmatika tidak tepat karena keterbatasan nilai eksponen atau karena rentang presisi.

2.4.4 *Overflow*

Pengecualian *overflow* mengisyaratkan ketika hasil perhitungan melebihi nilai maksimum yang dapat diwakili oleh format *floating point* standar IEEE 754. *Overflow* tidak aktif ketika salah satu *operand* adalah *infinity*. Pembagian dengan nol juga tidak mengaktifkan sinyal *overflow*.

2.4.5 *Underflow*

Pengecualian *underflow* mengisyaratkan ketika hasil dari perhitungan lebih kecil dari nilai minimum yang dapat diwakili oleh format *floating point* standar IEEE 754. Ada dua penyebab yang dapat mengaktifkan sinyal *underflow*, karena hasil perhitungan yang memang terlalu kecil atau karena proses normalisasi dan pembulatan.

Prosesor-prosesor sederhana umumnya hanya memiliki *unit aritmatika dan logika*, serta *unit kontrol* yang beroperasi pada dasar bilangan bulat (integer) saja. Untuk menghitung bilangan atau nilai *floating point* dapat dilakukan dengan bantuan software (perangkat lunak) sehingga operasinya menjadi lambat.

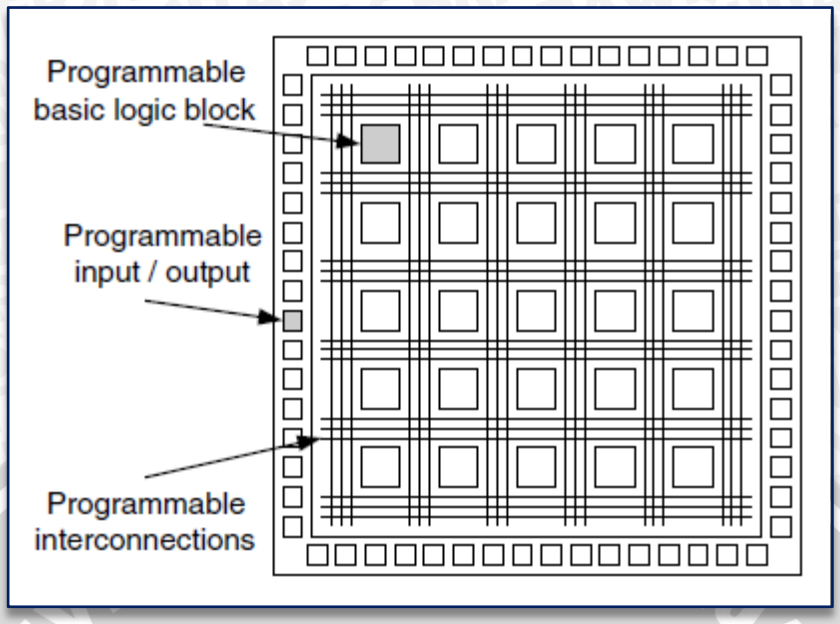
Melihat kenyataan ini sebenarnya diperlukan unit khusus atau sebuah prosesor tambahan yang digunakan untuk kalkulasi atau memproses bilangan floating point. Unit khusus untuk memproses bilangan floating point ini disebut *unit titik mengambang* atau *Floating Point Unit (FPU).*

2.5 *Field-Programmable Gate Array* (FPGA)

Field-Programmable Gate Array (FPGA) dapat digunakan untuk mengimplementasikan hampir semua desain hardware. Salah satu penggunaan umum dari FPGA adalah prototipe dari perangkat keras yang pada akhirnya akan dilaksanakan kemudian ke sebuah ASIC (*Application Spesific Integrated Circuit*). Namun demikian, FPGA telah semakin banyak digunakan sebagai platform produk akhir. Penggunaannya tergantung, untuk proyek tertentu, pada bobot relatif kinerja yang diinginkan, pengembangan, dan biaya produksi.

Mengapa FPGA? Sebuah Sejarah Survei Singkat Pada awal 1980-an, sebagian besar sistem rangkaian logika aplikasi khusus yang diimplementasikan dalam berbagai standar kecil *large scale integrated* (LSI) sirkuit: mikroprosesor, bus-I/O controller, sistem timer, dan sebagainya. Namun demikian, setiap sistem masih memiliki kebutuhan untuk acak "logika tempel" untuk menghubungkan IC yang lebih besar, misalnya, menghasilkan sinyal kontrol global dan format data (serial ke paralel, multiplexing, dll). IC buatan sering dirancang untuk menggantikan sejumlah besar logia tempel dan akibatnya mengurangi kompleksitas sistem dan biaya produksi, serta meningkatkan kinerja. Namun, IC buatan mahal untuk dikembangkan, sementara menghasilkan *time-to-market* (TTM) yang lama karena desain produksi yang mahal. Oleh karena itu pendekatan IC buatan hanya layak untuk produk dengan volume yang sangat tinggi (menurunkan dampak biaya NRE), dan tidak sensitif TTM. Mengatasi masalah ini, Xilinx™ (sebuah perusahaan rintisan) diperkenalkan, pada tahun 1984 FPGA sebagai teknologi alternatif IC buatan untuk menerapkan logika tempel. Berkat *computer-aided design* (CAD), sirkuit FPGA dapat diimplementasikan dalam waktu yang relatif singkat: tidak ada proses fisik tata letak, tidak memerlukan manufaktur IC, biaya NRE lebih rendah, dan TTM singkat.

Konsep dasar arsitektur FPGA terdiri dari *array* dua dimensi dari blok logika dan flip-flop tersusun sedemikian rupa agar pengguna dapat mengonfigurasi fungsi masing-masing blok logika, masukan dan keluaran, dan interkoneksi antar blok logika. Keluarga FPGA berbeda satu sama lain dalam cara pemrogramannya oleh pengguna, pengaturan kabel interkoneksi, dan fungsi dasar dari blok logika.



Gambar 2.3. Arsitektur dasar FPGA: *array* dua dimensi dari *programmable logic cells*, interkoneksi dan input/output.

Sumber: *Synthesis of Arithmetic Circuit: FPGA, ASIC, and Embedded Systems*.

Ada beberapa jenis metode pemrograman pada FPGA antara lain:

2.5.1 Berbasis SRAM

Sistem koneksi FPGA jenis ini menggunakan pass-transistor, gerbang transmisi, atau multiplekser yang dikendalikan oleh sel-sel SRAM. Teknologi ini memungkinkan rekonfigurasi sistem secara cepat.

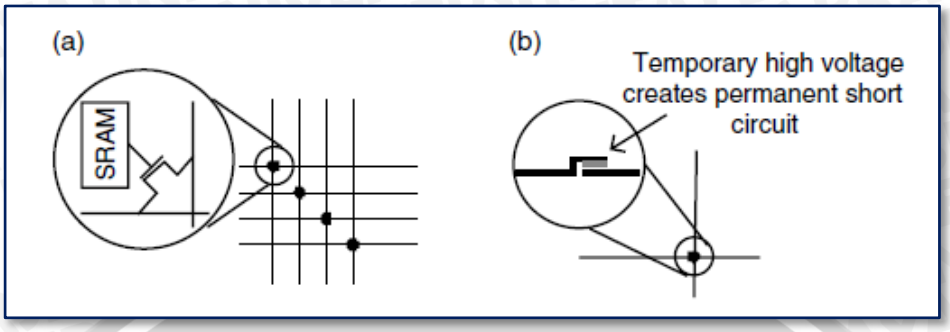
Kelemahan utama adalah ukuran chip, membutuhan teknologi yang dipakai pada RAM, dan membutuhkan beberapa sumber eksternal untuk memuat program yang akan diimplementasikan pada FPGA. FPGA dapat diprogram sebanyak tak terbatas.

Contoh keluarga FPGA yang menggunakan metode pemrograman jenis ini adalah Xilinx™ dan Altera™.

2.5.2 Teknologi antifuse

Suatu antifuse tetap dalam keadaan *high impedance* sampai diprogram menjadi *low impedance* atau "*fused*". Teknologi ini hanya dapat digunakan sekali pada perangkat *one-time programmable* (OTP) dan memiliki keunggulan lebih murah dari pada Teknologi RAM.

Contoh keluarga FPGA yang programmability-nya menggunakan teknologi antifuse adalah Actel™, Quicklogic™.



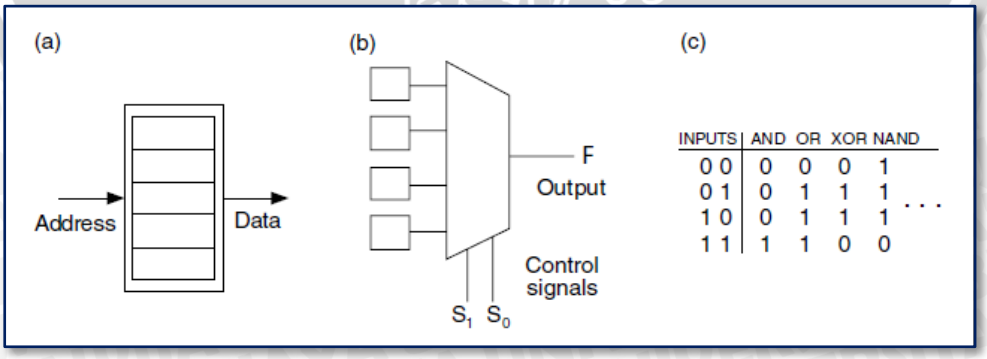
Gambar 2.4. Metode pemrograman (a) koneksi SRAM (b) antifuse
Sumber: *Synthesis of Arithmetic Circuit: FPGA, ASIC, and Embedded Systems*.

2.5.3 Teknologi EPROM/EEPROM

Metode ini sama dengan yang digunakan dalam penyimpanan EPROM/EEPROM. Konfigurasi sistem dapat tersimpan dalam satu chip FPGA atau tanpa membutuhkan memori eksternal. Umumnya FPGA jenis ini tidak memungkinkan untuk melakukan *in-chip programming*.

2.5.4 *Look-up Table*

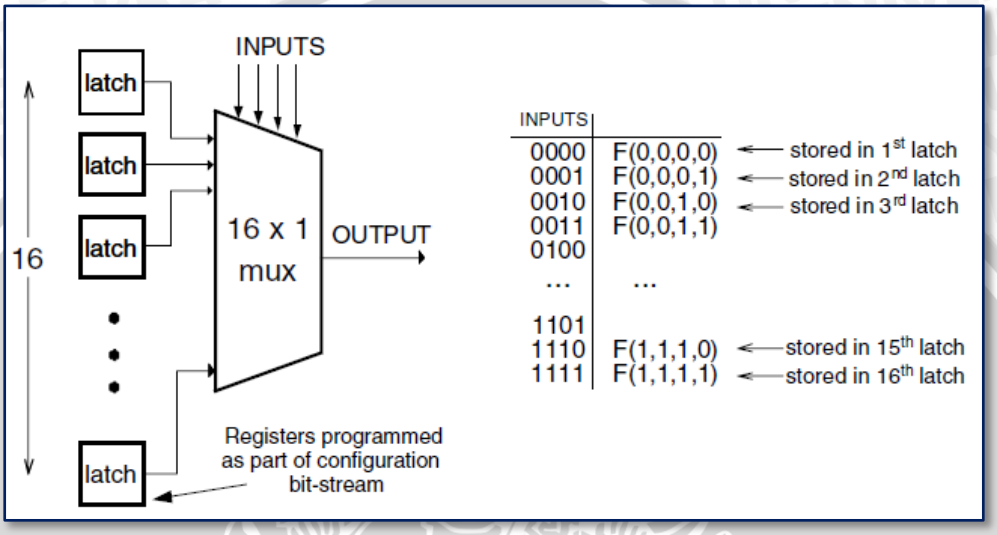
Look-up table adalah alternatif lain untuk mengimplementasikan fungsi logika kedalam FPGA. Blok logika yang melaksanakan fungsi logika adalah *look-up table* (LUT), diimplementasikan sebagai memori, atau multiplekser dan memori. Gambar2.5. menunjukkan alternatif ini, bersama dengan contoh isi memori untuk beberapa operasi dasar.



Gambar 2.5. *Look-up table* diimplementasikan sebagai (a) Memori. (b) Multiplekser dan Memori. (c) Contoh konten memori untuk beberapa fungsi logika.
Sumber: *Synthesis of Arithmetic Circuit: FPGA, ASIC, and Embedded Systems*.

Masukan kontrol multiplekser adalam masukan LUT, keluarannya adalah gerbang logika seperti pada umumnya. Sebuah LUT dapat diimplementasikan sembarang fungsi sebesar n-bit.

Sebuah n-LUT adalah implementasi langsung dari table kebenaran fungsi. Setiap *latch* menahan nilai fungsi sesuai dengan suatu kombinasi masukan. Sebuah contoh dari 4-LUT ditujukkan pada Gambar2.6.



Gambar 2.6. Implementasi 4-LUT dan isi tabel kebenarannya

Sumber: *Synthesis of Arithmetic Circuit: FPGA, ASIC, and Embedded Systems*.

Blok logika sederhana dalam FPGA dapat dirancang dengan menggunakan LUT, biasanya LUT 4-input, untuk mengimplementasi fungsi logika kombinasional, dan register yang secara opsional dapat digunakan untuk menyimpan output dari generator logika.