2. TEORI PENUNJANG

2.1. VHDL

VHDL (VHSIC *Hardware Description Language*) adalah Bahasa untuk mendeskripsikan *hardware* elektronika digital. VHDL ini salah satu HDL (*High Description Language*) yang populer. HDL bermacam-macam formatnya, antara lain yang banyak dipakai adalah VHDL, Verilog, dan Abel. Metode desain secara HDL ini memiliki kelebihan, dimana desain disimpan dalam bentuk *text* yang memudahkan penyimpanan. Dengan HDL, penterjemahan secara tradisional pendeskripsian desain ke persamaan logika dapat dieliminasi, hasil dari kode HDL diimplementasikan oleh perangkat *synthesis.*

Sebelum ditemukan HDL ini, dasar *systhesis design* dan implementasi menggunakan skematik yang diterjemahkan dalam persamaan *boolean* tertentu dari *gate* dan *flip-flop*, baik secara manual yang menggunakan *breadboard* atau perangkat CAE (*Computer Aided Enggineering*) yang menggunakan *programmable device*. Dengan kemampuan *device* sekarang, skematik mengalami keterbatasan karena IC yang terbaru terdiri dari jutaan *gate*s dengan densitas yang terus berkembang. Dalam *design system digital* yang besar, banyak waktu rekayasa terbuang dalam mengubah-ubah format menggunakan bermacam-macam alat bantu desain dan simulator. Dengan VHDL, hal ini tidak diperlukan lagi.

Sejarah penemuan HDL dimulai saat pencarian program alat Bantu dokumentasi dan desain standar untuk VHSIC (*Very High Speed Integrated Circuits*) oleh Departemen Pertahanan Amerika Serikat (*United States Departement of Defence*, DoD). DoD pada musim panas 1981 mensponsori *workshop Hardware Description Languages* (HDL) di Woods Hole, Massachussets. *Workshop* ini diatur oleh *Institute for Defence Analysis* (IDA) untuk mempelajari bermacam-macam metode deskripsi perangkat keras, keperluan akan bahasa standar dan cirri-ciri yang diperlukan.

Pada tahun 1983, DoD menetapkan perlunya standar untuk VHSIC *Hardware Description Language* (VHDL) berdasar rekomendasi yang diberikan oleh *workshop* di Woods Hole tersebut. Kontrak pengembangan bahasa VHDL, lingkungan dan perangkat lunaknya diberikan kepada IBM, Texas Instruments,

dan Intermetrics Corporations. Pengembangan ini dimulai pada musim panas 1983.

VHDL versi 2.0 dirilis enam bulan kemudian, dimana perangkat lunak ini mempunyai kekurangan yang mengganggu desain tingkat lanjut. Kekurangan ini dihilangkan pada VHDL versi 6.0 yang dirilis pada Desember 1984. Alat Bantu berbasis VHDL juga mulai dikembangkan pada tahun 1984.

Pada tahun 1985, hak cipta VHDL versi 7.2 ditransfer kepada IEEE untuk pengembangan lebih lanjut dan standarisasi. Ini mengarah ke pengembangan Referensi Manual Bahasa VHDL IEEE 1074/A, yang dirilis bulan Mei 1987. Pada Desember 1987, VHDL 1076-1987 secara formal menjadi standar IEEE untuk HDL.

2.1.1. Standar IEEE 1076 dan Standar IEEE 1164

IEEE Standard VHDL Language Reference Manual, Std 1076-1993, IEEE, NY, 1993. Standar ini merupakan buku pedoman resmi dari bahasa VHDL. Standar ini menjadi buku pegangan dalam penulisan mulai dari tingkat pemula hingga tingkat mahir.

Untuk mengatasi standarisasi tipe data, dikembangkan IEEE Standard Multivalue Logic for VHDL. Model *Interoperability* (STD_Logic_1164), Std1164-1993, IEEE, 1993. Standars ini berisi definisi dari standard *nine-valued data type*. Standard data ini disebut std_logic, dan IEEE 1164 package biasanya diacukan sebagai *standard_logic_package*.

Standar 1076.3 (biasanya disebut *Numeric Standard* atau *Synthesis Standard*) menetapkan paket standar dan interpretasi untuk tipe data VHDL yang dihubungkan dengan *hardware* yang sebenarnya. *Standard* ini diperbaiki pada akhir 1995 diunggulkan dapat mengganti *non standard packages* yang diciptakan oleh bermacam-macam perusahaan yang pada umumnya disertai dengan barang produksinya.

2.1.2. Proses Desain VHDL

Untuk spefikasi desain, VHDL digunakan paling awal (saat masih mendesain pada *high-level*) untuk mengambil performa dan *interface* setiap

komponen yang diperlukan pada sistem yang besar. Ini merupakan keuntungan untuk proyek besar sehingga dapat diselesaikan oleh beberapa anggota tim. Metode VHDL menggunakan *top-down approach* untuk mendesain. *Desainer system* mungkin menetapkan *interface* untuk setiap komponen pada *system* dan mendeskripsikan kebutuhan yang dapat diterima pada komponen itu dalam *high level test bench*.

Definisi *interface* (terekspresikan sebagai deklarasi *entity* VHDL) dan spesifikasi performa *high-level* (biasanya pada *test bench*) dapat diberikan pada anggota tim lain untuk kemudahan penyelesaian. *Desain capture* adalah fase dimana detail desain diberikan (*captured*) pada komputer desain. Pada desain *entry* (*capture*), desain itu diekspresikan sebagai skematik atau menggunakan deskripsi VHDL.

Untuk simulasi desain (*verify*), setelah desain dimasukkan dalam komputer maka operasi rangkaian itu disimulasikan untuk melihat hasil dari fungsi yang telah ditentukan. Pertama harus membuat satu atau lebih *test bench* sebagai bagian dari spesifikasi desain. Berikutnya menggunakan *simulator* untuk memberikan kepercayaan bahwa desain itu telah beroperasi seperti yang telah diharapkan.

Pada bagian akhir dilakukan simulasi sekali lagi terhadap data yang dihasilkan oleh hasil implementasinya.

2.1.3. Penulisan Desain dalam *syntax* VHDL

Komponen dasar pembentuk dari *entity* vhdl adalah *entity declaration* dan *architecture body*. *Desain entity* merupakan abstraksi dari suatu desain yang menggambarkan suatu system secara lengkap. *Entity declaration* menggambarkan desain I/O sedangakan *architecture body* menggambarkan fungsi dari desain *entity. Entity declaration* analog dengan skematik symbol yang menjelaskan hubungan komponen-komponen dalam suatu desain. *Signa*l I/O di *entity declaration* didefinisikan sebagai *port* yang analog dengan *pin* dari *symbol* skematik.

Struktur penulisan *entity declaration* harus mengandung *unsure name* dan *port declaration*, pada *port declaration* harus mengandung *unsure name (identifier), direction (mode)* dan tipe data. Nama di *entity declaration*

(menjelaskan jenis sistem digital dari suatu desain), nama *port declaration* (menjelaskan nama-nama *pin* I/O sistem digital dari suatu desain), *direction mode* (menjelaskan kondisi I/O dan tipe data menjelaskan tipe data yang dipakai), sesuai dengan standar IEEE 1076/1164. Ada 4 macam tipe data, yaitu IN, OUT, BUFFER dan INOUT. Ada dua macam tipe data yang digunakan berdasarkan standar IEEE, yaitu IEEE 1076/93 (Boolean, Bit, Bit_Vector, dan integer) dan IEEE 1164 (STD_Ulogic, STD_logic dan STD_Logic_Vector).

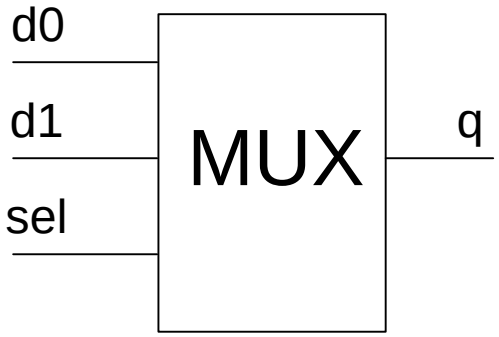
Architecture body berhubungan dengan *Entity declaration* di dalam desain *entity* VHDL, *architecture body* menggambarkan isi dari *entity* yang menjelaskan fungsi dari *entity*. Ada beberapa metode dalam penulisan *architecture body*, yaitu:

1. Structural Description

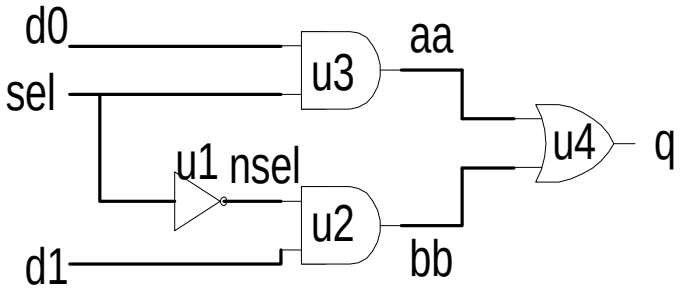
Metode ini sama dengan representasi skematik, karena hubungan tiap komponen diperlihatkan. Cara ini kurang memberikan manfaat, terutama bila sistem yang didesain sudah kompleks.

Contoh:

Rangkaian multiplekser dua *input*.



Gambar 2.1. Diagram Blok *Mux* 2 *Input*



Gambar 2.2. Rangkaian Skema *Mux* 2 *Input*

Deskripsi VHDL-nya:

```
ENTITY mux IS
PORT (d0,d1,sel : IN BIT;
        q           : OUT BIT)
END mux;

ARCHITECTURE struc OF mux IS
COMPONENT and2
PORT ( a,b : IN BIT;
        c     : OUT BIT)
END COMPONENT;

COMPONENT or2
PORT ( a,b : IN BIT;
        c     : OUT BIT)
END COMPONENT;

COMPONENT inv
PORT ( a : IN BIT;
        C  : OUT BIT)
END COMPONENT;

SIGNAL (aa,bb,nsel : BIT);
FOR u1 : inv USE ENTITY WORK inv(dftl);
FOR u2, u3 : and2 USE ENTITY WORK and2(dftl);
FOR u4 : or2 USE ENTITY WORK or2(dftl);
BEGIN
u1 : inv PORT MAP (sel,nsel);
u2 : and2 PORT MAP (nsel, d1, bb);
u3 : and2 PORT MAP (sel, d0, aa);
u4 : or2 PORT MAP (aa, bb, q );
END struc;
```

2. Data Flow Description

Deskripsi ini sama dengan bahasa transfer register (*register transfer languge*), Dimana fungsi dari rangkaian dijelaskan dengan mendefinisikan aliran informasi dari satu *input* atau register ke register lain atau output.

Contoh:

Rangkaian multiplekser dua *input*.

Deskripsi VHDL-nya adalah:

```
ENTITY mux IS
PORT (d0,d1,sel : IN BIT;
        q           : OUT BIT)
END mux;
```

```
ARCHITECTURE data_flow OF mux is
BEGIN
cs1: q <= di WHEN sel = '0' ELSE d0;
END data_flow;
```

3. Behavioral Description

Deskripsi ini menjelaskan tingkah laku (*behavior*) fungsional rangkaian yang dirancang, serta respon rangkaian terhadap berbagai *signal* masukan. Tingkah laku rangkaian diterangkan secara algoritmik tanpa memperlihatkan bagaimana strukturnya diimplementasikan.

Contoh:

Rangkaian multiplekser dua *input.*

Deskripsi VHDL-nya adalah:

```
ENTITY mux IS
PORT (d0,d1,sel : IN BIT;
        q       : OUT BIT)
END mux;


ARCHITECTURE behav OF mux IS
BEGIN
f1 :  PROCESS (d0, d1, sel)
BEGIN
IF sel='0' THEN
        q <= d1;
ELSE
        q <= d0;
END IF;
END PROCESS f1;
END behav;
```

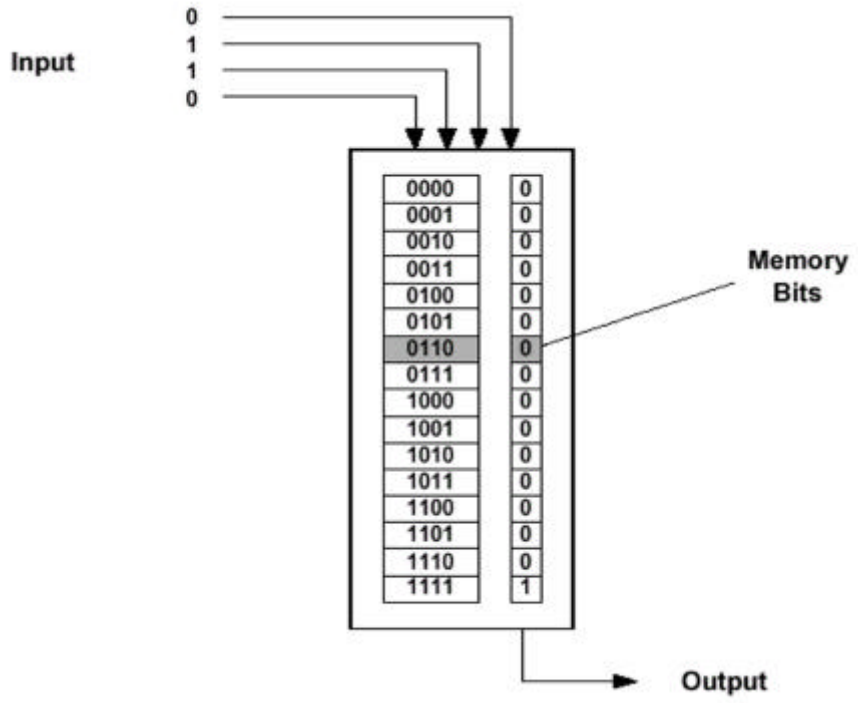
2.2. Xilinx XC4005XL

Implementasi menggunakan XC4005XL-PC84c-3, yang merupakan 84 pin J-lead PLCC (*Plastic-leaded chip carrier*) *package*. XC4005XL merupakan keluarga XC4000E dan XC4000X dari Xilinx, memiliki 468 logic cells, 14 x 14 =196 CLB, 616 Flip-flops dan 64 I/O yang tergabung dalam IOB.

Berbeda dengan komponen tradisional lainnya, PAL, EPLD, dan *gate array*, Xilinx FPGA mengimplementasikan *logic* kombinasional (AND, OR, dsb) ke dalam *Look Up Table* kecil (16 x 1 ROMs). FPGA terdiri dari *block-block*,

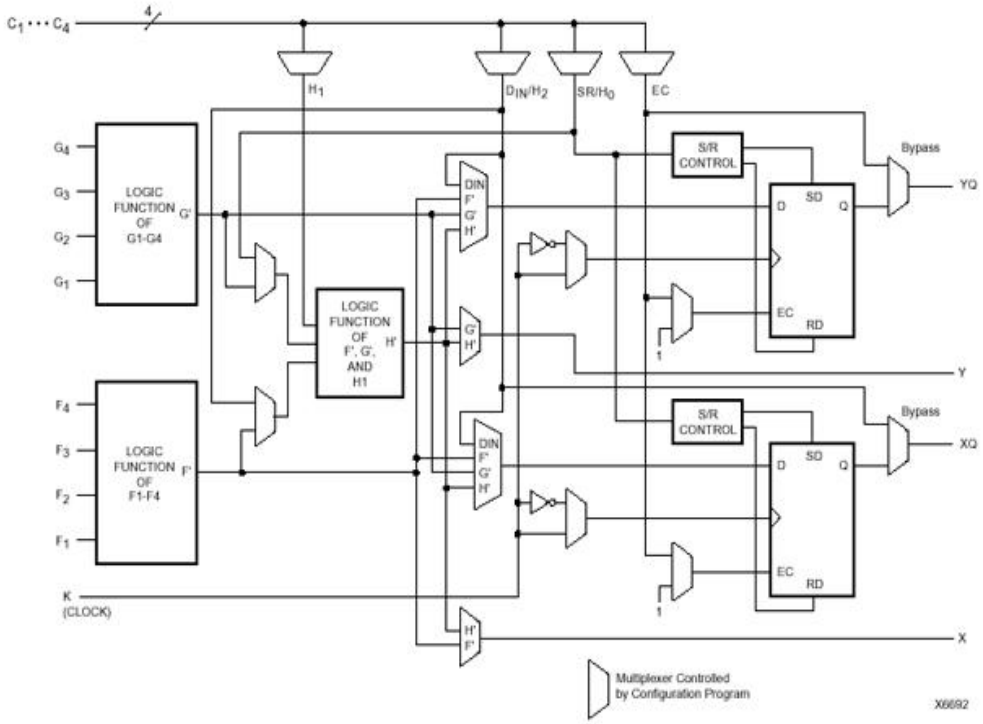
dimana *Look Up Table* (LUT) komponen utama arsitektur FPGA. Keadaan tipikal dari LUT memiliki hanya 4 *input* dan memory kecil berisi 16 bit (ROM 16 x 1). Bila *input* diberi kombinasi misalnya 0110, menunjuk pada alamat pada ROM dan isi dari ROM tersebut dikeluarkan. Semua logic *input* 4 bit dapat disusun dengan memrogram LUT. Contohnya AND 4 *input* dibuat dengan mengisi alamat ROM dengan 0, kecuali pada alamat "1111" dengan 1, dimana komponen ini aktif jika semua *input* 1.

Setiap CLB (*Configurable Logic Block*) memiliki tiga *look-up tables* (LUTs) dan dua *flip-flops*. Tiap CLB dapat dipakai sebagai sembarang *logic* dengan 4 *input* atau 16 x 1 bit *synchronous static* RAM atau ROM. Juga punya "*carry logic*" guna membentuk *ripple carry adder* yang cepat dan kompak. IOB (I/O *Block*) terdiri dari *input* dan *output buffer* dan *flip-flop*. *Output buffer* dapat berfungsi *three stated* untuk *bi-directional* I/O. *Programmable interconnect* atau *Programmable Switch Matrix* (PSM) menghubungkan CLB / IOB pada *input* CLB / IOB lainnya. Kebanyakan FPGA memiliki lebih banyak CLB daripada pin I/O, jadi tiap CLB tidak dapat langsung berhubungan dengan dunia di luar FPGA. Ini berbeda dengan SPLD atau CPLD.



Gambar 2.3. LUT 4 *Input* Sebagai AND *Gate*

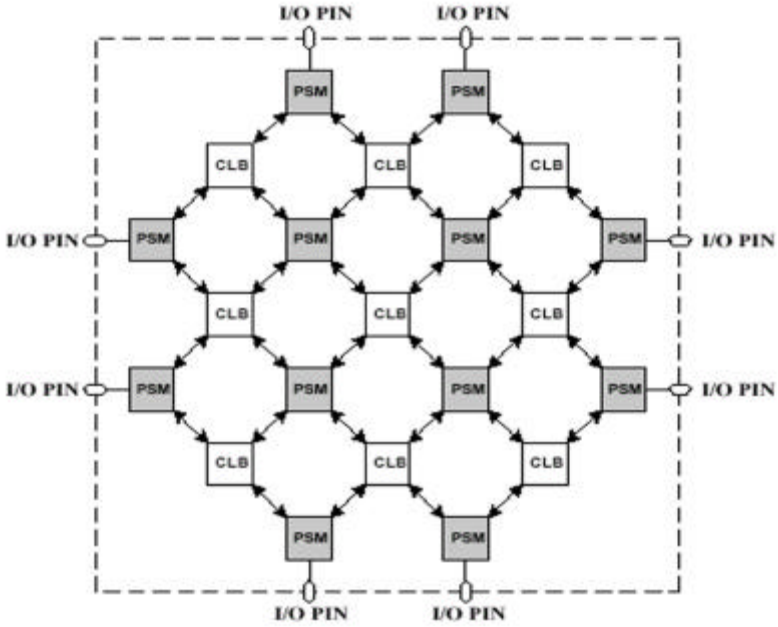
Sumber: *The Programmable Logic Data Book*. San Jose: Xilinx, 1999. p.23.



Gambar 2.4. CLB

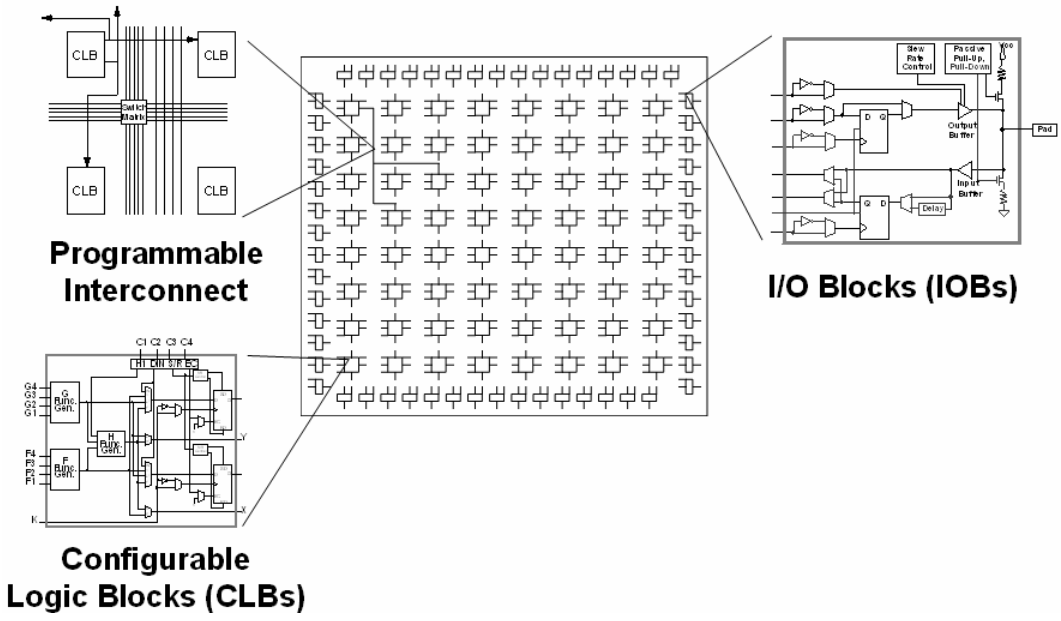
Sumber: *The Programmable Logic Data Book*. San Jose: Xilinx, 1999. p.31.

Hubungan antara CLB, PSM dan IOB diatur secara *electrical*. Produk awal PLD memakai *switch array* dari *fuse* (sekering) pada tiap titik pertemuan (*crosspoint*). Di titik ini ditempatkan rangkaian spesial yang dapat membakar *fuse* dengan memberi tegangan pemrograman yang lebih tingi. Teknologi ini menyebabkan perangkat hanya dapat dipakai sekali saja.



Gambar 2.5. Arsitektur Umum FPGA

Sumber: *The Programmable Logic Data Book*. San Jose: Xilinx, 1999. p.26.



Gambar 2.6. Arsitektur Xilinx FPGA

Sumber: *The Programmable Logic Data Book*. San Jose: Xilinx, 1999. p.26.

Kedua gambar diatas merupakan gambaran umum di dalam FPGA. Terdapat CLB (*Configurable Logic Block*), PSM (*Progammable Switch Matrix)* atau *Progammable Interconnect* yang berfungsi sebagai rute yang menghubungkan antara CLB dengan CLB dan I/O *Blocks*.

Keunggulan memakai XC4000XL Family adalah:

- *Low voltage* (3.3 Volt), berdasarkan teknologi SRAM, 5 Volt *tolerant* I/O.
- Kecepatan tinggi (80 MHz).
- Arus 12 mA sink per *output*.
- *On Chip Oscilator* (sampai 8 MHz).
- *High capacity*.
- *On-chip ultra fast RAM* dengan *synchronous write* dan *dual-port RAM*.
- Rekonfigurasi tanpa batas (*bitstream*).
- Mudah diubah tergantung aplikasi.
- Terdiri dari CLB, IOB dan PSM.

Harga dari FPGA pada umumnya cukup mahal, oleh karena itu penggunaan FPGA lebih ditujukan untuk proses pembelajaran atau pembuatan suatu sistem elektronika. Setelah sistem berhasil dibuat maka dapat digunakan dalam produksi, sistem tersebut dimasukkan/ditulis ke dalam IC yang diproduksi secara massal. Dengan adanya FPGA yang memiliki kemampuan untuk dikonfigurasi berulang kali sangat bermanfaat dalam menghemat biaya *prototype* suatu IC.

Beberapa contoh aplikasi dari penggunaan FPGA antara lain adalah:

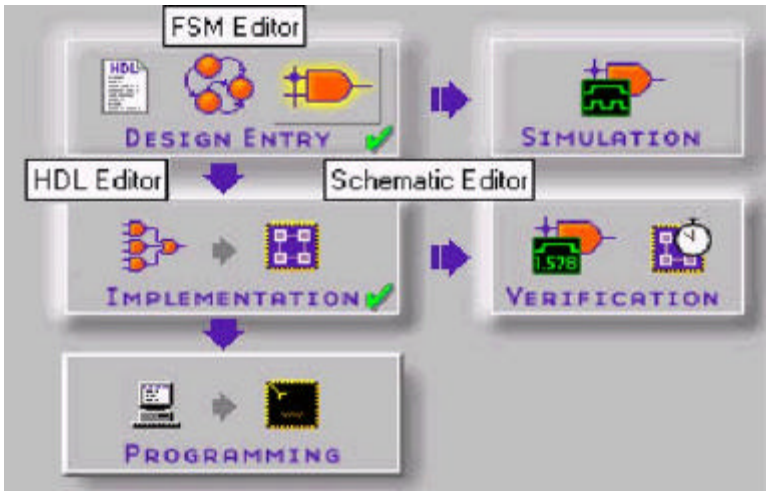
- ✓ FPGA sebagai ALU *microcontroller*
- ✓ FPGA untuk membaca kapasitas memori dari VGA
- ✓ FPGA sebagai *calculator*
- ✓ FPGA sebagai penunjuk waktu (jam dan tanggal) dari RTC

2.3. Xilinx Foundation 2.1

Dalam *Foundation* dapat dipilih desain dengan *schematic* atau HDL. Pada desain ini memakai *schematic*, dalam skematik desain dapat memakai tiga buah metode, dimana ketiga-tiganya dapat saling berinteraksi :

1. *Schematic*, memakai komponen yang ada pada *library* dan kedua deskripsi lainnya.
2. *State Diagram*, desain dimasukkan dengan menggambarkan interaksi antara *state*.
3. HDL, desain dideskripsikan dengan kode HDL, dipakai VHDL.

Setelah proses desain selesai, kemudian dilanjutkan simulasi dengan *timing* diagram untuk melihat kebenaran algoritma yang dihasilkan. Setelah itu desain diimplementasikan dengan bantuan XSTOOL ke XS4005XL *board*.

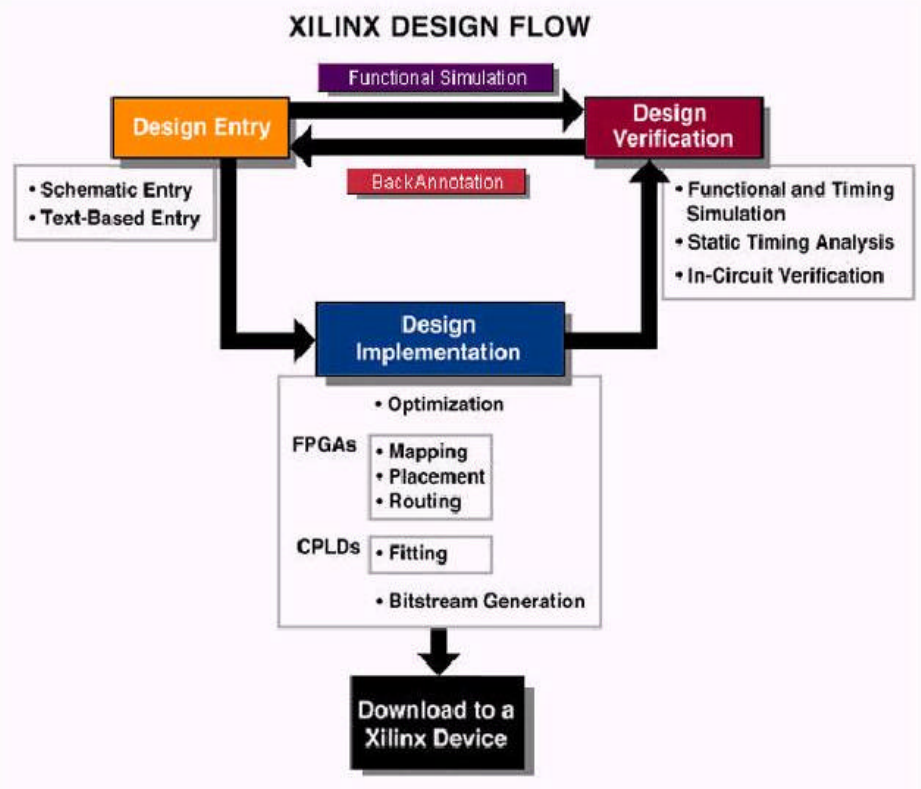


Gambar 2.7. *Flow* Skematik

Untuk memulai desain dengan *Foundation* dan mengimplementasikan ke dalam *chip* harus dilakukan beberapa tahap sebagai berikut ini:

1. Mendapatkan ide, berupa gambaran bagaimana suatu system bekerja.
2. Definisikan *input* dan *output*, diperlukan dalam desain *top-down*.
3. Masukan desain dalam Foundation, dapat melalui *schematic editor*, *state machine editor* atau HDL *editor*, dan bisa campuran dari ketiganya.
4. Simulasi secara fungsional, dengan *timing* diagram dan memperbaiki kesalahan yang terjadi dalam desain.
5. Implementasi desain, dalam hal ini ke FPGA dimana memerlukan beberapa langkah yaitu *mapping* ke dalam arsitektur FPGA, menempatkan *gate*s ke dalam CLB dan menghubungkan koneksi dengan PSM.
6. *Download* ke XS4005XL *board*, memakai XSTOOL untuk *download file bitstream* yang dihasilkan ke dalam *board*.

Tahap 1 dan 2 dilakukan di atas kertas. Tahap 3 sampai 5 dilakukan dengan Xilinx Foundation 2.1, dan tahap 6 memakai XSTOOLS pada XS-Board.



Gambar 2.8. *Design Flow*

Dengan memakai skematik, maka penggambaran tiap komponen penunjang diubah menjadi *netlist* ".xnf", termasuk pendeskripsikan secara HDL, setelah itu dipanggil dalam skema utama menjadi salah satu komponen dari *library*. Dalam skema utama digambar secara lengkap hubungan tiap komponen. Kemudian hasil dari desain diubah ke *netlist* ".xnf", setelah itu baru diimplementasikan yang menghasilkan *file bitstream* ".bit".

Setelah proses implementasi dilaporkan beberapa *file report*, antara lain:

1. *Translation Report* ".bld", *file* ini melaporkan *problem* yang terjadi saat *netlist* diubah ke dalam format *internal* (yang dipakai *Foundation Implementation tools)*. *Design Rule Check* (mengecek format desain) dan *user constrain file* ".ucf" dicek kebenarannya.
2. *Map Report* ".mrp", melaporkan tipe optimasi pada *netlist*, *logic gates* yang tidak terpakai dibuang dengan optimasi dengan tanpa mengganti fungsinya. Juga menunjukkan bagaimana *logic gates* dikelompokkan dan dipetakan dalam LUT dari CLB pada FPGA.

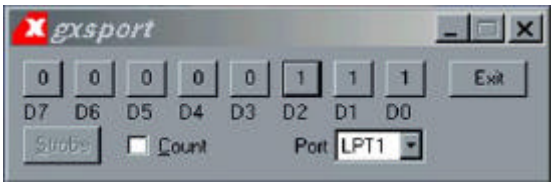
3. *Place & Route Report* ".par", melaporkan isi catatan tentang besar isi dari FPGA yang terpakai, juga menunjukkan keinginan pengguna dalam implementasi.
4. *Pad Report* ".pad", melaporkan nomor *pin* terminal I/O yang digunakan pada *device*.
5. *Asynchronous Delay Report* ".dly", melaporkan propagansi waktu tunda (*delay*) dari setiap sinyal.
6. *Post Layout Timing Report* ".twr", melaporkan semua kemungkinan yang tidak dapat dilaksanakan dalam *placed & route logic* yang mengganggu timing.
7. *Bitgen Report* ".bgn", melaporkan keberhasilan pembuatan file *bitstream*.

2.4. XSTOOLS

XSTOOLS merupakan perangkat lunak yang digunakan untuk men-*download bitstream* dan *hex code* yang telah dirancang. Dilakukan dengan memanggil GXSLOAD dan meletakkan *file bitstream* yang telah dibuat ke dalam window GXSLOAD. Terdapat GXSPORT untuk membantu proses *debug*, dimana D7 sampai D0 dapat menjadi *input port* ke dalam XS *board*.

Gambar 2.9. GXSLOAD

Gambar 2.10. XS *Board Test*



Gambar 2.11. GXSPORT

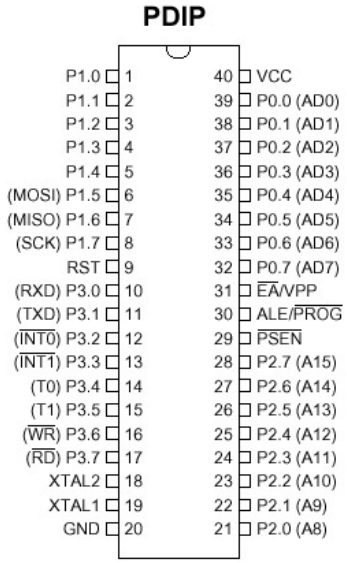
2.5. *Microcontroller* AT89S51

AT89S51 adalah *microcontroller* yang hemat energi dengan kemampuan yang tinggi dari CMOS 8 *bit*, dengan 4 *kilobyte* dari *In-System Programmable Flash Memory*. Alat ini diproduksi menggunakan *Atmel high density nonvolatile memory technology* dan cocok dengan cara penggunaan dan kaki – kaki keluaran (*pin out*) dari industri standar 80C51. *On-chip* yang cepat (*flash*) memperbolehkan program untuk diprogram ulang oleh sistem dalam atau oleh pemograman memori tetap konvensional. Dengan menggabungkan sebuah CPU 8 bit dengan *flash* pada *chip* tunggal, maka Atmel 89S51 adalah *microcontroller* yang berguna untuk menyediakan pemecahan yang fleksibel dan harga yang efektif untuk banyak aplikasi kontrol.

AT89S51 menyediakan sifat-sifat standar sebagai berikut : 4 *kilobyte flash*, 128 *byte* RAM, 32 jalur masukan / keluaran (I/O), *watchdog timer*, dua data *pointer*, dua buah penghitung waktu / penghitung 16 *bit*, sebuah *vector* dua *level interrupt* arsitektur, *full duplex serial port*, *on chip oscillator*, dan *clock circuitry*. Sebagai tambahan, AT89S51 di-design beroperasi hingga frekuensi 20 MHz dan menyediakan dua *software* mode penghematan tenaga. *Mode Idle* menghentikan CPU sementara memperbolehkan RAM, penghitung waktu/penghitung *port* serial dan sistem interupsi untuk tetap berfungsi. *Mode Power Down* menyimpan isi RAM tetapi membekukan *oscillator* yang menyebabkan semua fungsi *chip* lainnya tidak berfungsi sampai *reset hardware* berikutnya.

2.5.1. Deskripsi *Pin*

Microcontroller AT89S51 memiliki 40 *pin*, 32 *pin* diantaranya adalah *directional* I/O yang terbagi dalam 4 *port*. Konfigurasi dari *pin-pin* tersebut, yaitu:



Gambar 2.12. Konfigurasi *Pin* AT89S51

Sumber: *AT89S51 Datasheet*, (*AT89S51.pdf)*. San Jose: Atmel Corporation, 2003. p.2.

- V_{cc} (*pin* 40), merupakan *pin supply* tegangan sebesar +5 Volt.
- GND(*pin* 20), merupakan *pin* tegangan referensi 0 Volt (*ground*).
- RST (*pin* 9), sebagai masukan *reset*, yaitu jika pada saat diberi tegangan +5 Volt, maka seluruh isi dari *internal memory* dan *register-register* yang dimiliki AT89S51 akan kembali ke kondisi *reset.*
- EA – *External Access* (*pin* 31), pada waktu *pin* ini diberi tegangan +5 Volt, maka AT89S51 akan mengeksekusi program *internal*. Dan sebaliknya jika *pin* ini diberi referensi tegangan 0 Volt (*ground*), maka AT89S51 akan berada dalam *mode access external* ROM yang mulai dari alamat 0000_H-$FFFF_H$.
- XTAL1 (*pin* 19), sebagai masukan ke *inverting oscillator amplifier* dan masukan ke *internal clock operating circuit.*
- XTAL2 (*pin* 18), sebagai keluaran dari *inverting oscillator amplifier*.
- PSEN – *Programmable Strobe Enable* (*pin* 29), merupakan sinyal yang dikeluarkan oleh AT89S51 untuk membaca *external program memory* (*fetching*).

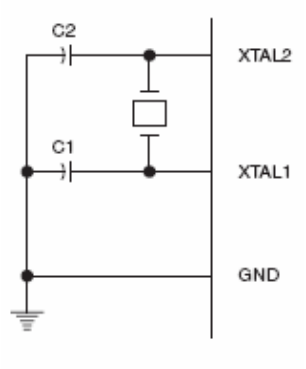
- ALE – *Address Latch Enable* (*pin* 30), Suatu keluaran sinyal yang berfungsi untuk memisahkan *address bus byte* rendah (A_7 – A_0) yang sebelumnya di multipleks dengan *data bus* dalam AD_7 – AD_0 selama mengakses *external memory*. Sinyal ini berupa pulsa persegi yang keluar terus-menerus dengan frekuensi 1/6 dari frekuensi kristal. ALE akan aktif selama perintah MOVX atau MOVC.
- *Port* 0 (*pin* 32, 33, 34, 35, 36, 37, 38, 39), *Port* ini dapat digunakan sebagai I/O dua arah yang dapat diakses per *bit* dengan menambahkan *pull-up* resistor. *Port* ini juga berfungsi sebagai *address bus* byte rendah (A_7 – A_0) dan *data bus* (D_7 – D_0) yang didesain secara multipleks (sehingga *port* ini diberi nama AD_7 – AD_0).
- *Port* 1 (*pin* 8, 7 , 6, 5, 4, 3, 2, 1), digunakan sebagai I/O dua arah yang dapat diakses per *bit* dengan *internal pull-up*.
- *Port* 2 (*pin* 28, 27, 26, 25, 24, 23, 22, 21), *Port* ini dapat digunakan sebagai I/O dua arah yang dapat diakses per *bit* tanpa menambahkan *pull-up* resistor karena terdapat *internal pull-up.* Selain itu *Port* ini berfungsi sebagai *address bus byte* tinggi(A_{15} – A_8).
- *Port* 3 (*pin* 17, 16, 15, 14, 13, 12, 11, 10), digunakan sebagai I/O dua arah yang dapat diakses per *bit* dengan *internal pull-up.* Selain itu *Port* 3 juga mempunyai fitur-fitur spesial yang dapat dilihat pada tabel berikut ini :

Tabel 2.1. *Port* 3 AT89S51

Port Pin	**Fungsi Alternatif**
P3.0	RXD (serial *input* port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (timer 0 exterbal *input*)
P3.5	T1 (timer 1 external *input*)
P3.6	WR (external data memory write strobe)
P3.7	RD (external data memory read strobe)

Sumber: *AT89S51 Datasheet*, (*AT89S51.pdf)*. San Jose: Atmel Corporation, 2003. p.5.

Hal yang berhubungan dengan IC AT89S51 ini adalah rangkaian kristal dan rangkaian *reset*. Rangkaian kristal terdiri dari 2 buah kapasitor keramik berukuran 33pF dan sebuah kristal 11,0592 MHz. Rangkaian kristal ini dihubungkan pada IC AT89S51 pada pin 19 (XTAL1) dan pin18 (XTAL2) yang dapat dilihat pada gambar 2.2.

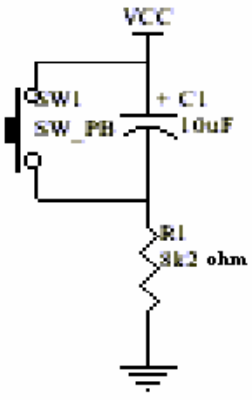


Gambar 2.13. Rangkaian Kristal

Sumber: *AT89S51 Datasheet*, (*AT89S51.pdf*). San Jose: Atmel Corporation, 2003. p.2-35.

Rangkaian *Power on Reset* berfungsi untuk mengembalikan nilai-nilai *register* yang ada pada IC AT89S51 ke dalam kondisi mula-mula, dan akan menyebabkan *microcontroller* menuju ke alamat 0000_H. Rangkaian ini terdiri dari:

- 1 buah kapasitor 10µF
- 1 buah resistor 8,2 K?
- 1 buah SW-PB



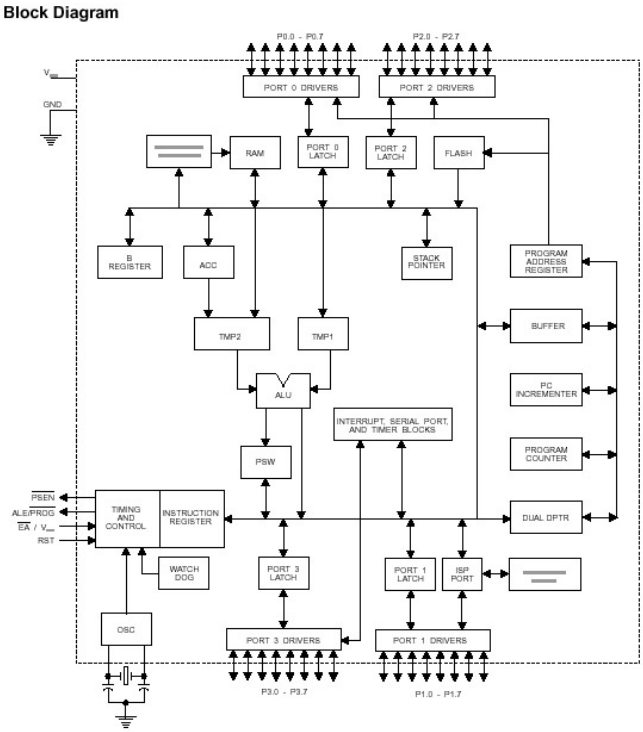
Gambar 2.14. Rangkaian *Power on Reset*

Sumber: *AT89S51 Datasheet*, (*AT89S51.pdf*). San Jose: Atmel Corporation, 2003. p.2-35.

Cara kerja dari rangkaian ini adalah pada saat rangkaian diberi tegangan pertama kali, maka arus akan mengalir pada kapasitor (yang semula *short circuit*, perlahan-lahan mulai terisi) dan pada titik *pin reset* yang dihubungkan ke *microcontroller* akan mendapatkan tegangan Vcc (logika ‘1’) sehingga *microcontroller* dalam kondisi *reset*. Beberapa saat kemudian isi dari kapasitor akan penuh dan menyebabkan tidak ada arus yang mengalir pada kapasitor (sehingga seolah-olah *open circuit*). Titik *pin reset* akan mendapat tegangan setara GND (yang menyebabkan *microcontroller* aktif kembali). Fungsi resistor pada rangkaian ini adalah untuk mengatur lama waktu pengisian tegangan pada kapasitor. Nilai resistor tersebut didapat dari *datasheet*. Pada rangkaian ini, EA=’1’ (karena setiap pengaksesan program memory berada dalam internal ROM).

2.5.2. Diagram Blok

Diagram blok dari *microcontroller* AT89S51 ditunjukkan pada gambar berikut ini:

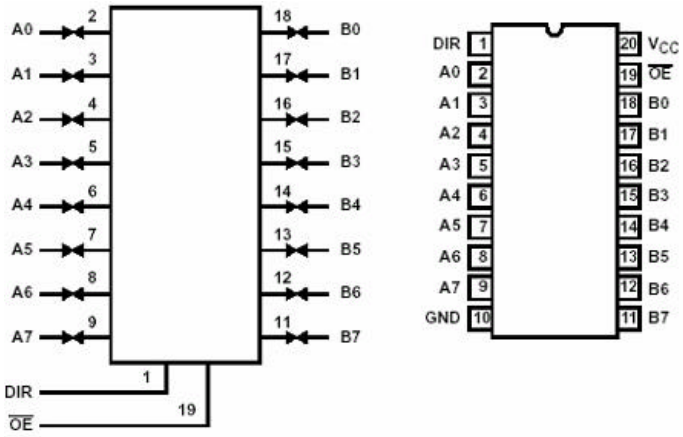


Gambar 2.15. Arsitektur *Microcontroller* AT89S51

Sumber: *AT89S51 Datasheet*, (*AT89S51.pdf)*. San Jose: Atmel Corporation, 2003. p.2.

2.6. 74HCT245

74HCT245 berfungsi sebagai buffer untuk komunikasi dua arah yang dapat diatur aliran data 8 bit-nya. IC ini memiliki output yang mendekati sumber tegangan atau mendekati tegangan 5 volt.



Gambar 2.16. Diagram Fungsional 74HCT245

Sumber: 74HCT245 *Datasheet*, (74HCT245.*pdf*). Texas: Texas Instruments, 1997. p.2.

Arah aliran diatur oleh pin DIR, nilai logic yang masuk ke pin DIR menyebabkan aliran data dapat dari A ke B atau B ke A. Berikut Deskripsi *Pin* dari 74HCT245 :

- VCC : sumber tegangan.
- GND : *grounding* tegangan.
- A0 – A7 : data *bus, bus lines bidirectional*.
- B0 – B7 : data *bus, bus lines bidirectional*.
- OE : enable IC, saat *logic low* maka IC akan aktif.
- DIR : pin untuk mengatur arah aliran data.

Tabel 2.2. Tabel Kebenaran 74HCT245.

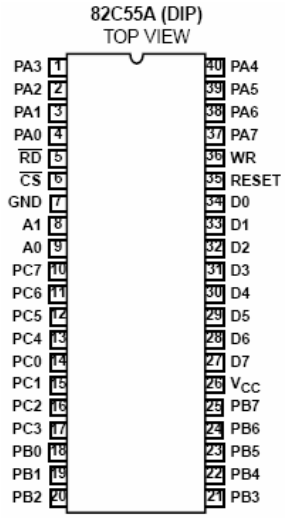
CONTROL INPUTS		
$\overline{OE}$	DIR	OPERATION
L	L	B Data to A Bus
L	H	A Data to B Bus
H	X	Isolation

Sumber: 74HCT245 *Datasheet*, (74HCT245.*pdf*). Texas: Texas Instruments,1997. p.2.

2.7. PPI 8255

IC PPI 8255 merupakan IC yang berfungsi sebagai ekspansi *port*, PPI 8255 dapat digunakan dengan berbagai macam jenis *microprocessor*. IC ini memiliki 3 *port input/output* 8-bit. *Port-port output* ini dinamakan *Port-A, Port-B,* dan *Port-C*. PPI 8255 dapat dioperasikan dalam 3 macam mode, yaitu mode 0, 1, dan 2.

2.7.1. Deskripsi *pin*



Gambar 2.17. IC PPI 8255

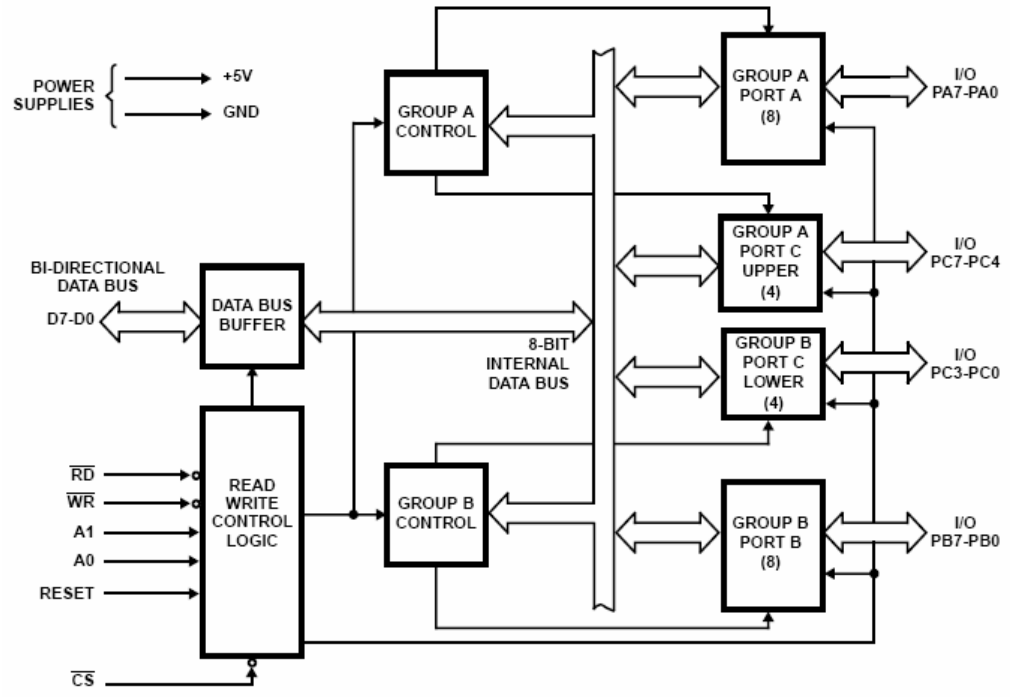
Sumber: 82C55A *Datasheet*. (82C55A.*pdf*). San Jose: Harris Semiconductor, 1996. p.1.

- VCC : sumber tegangan
- GND : *grounding* tegangan
- D0 – D7 : data *bus, bus lines bidirectional* yang berhubungan langsung dengan system data bus di dalamnya.
- Reset : *logic high* pada pin ini akan menyebabkan semua *port* di-*set* ke *mode input.*
- CS : pin untuk mengaktifkan IC, *active low.*
- RD : berfungsi sebagai pengontrol kondisi status *read*, bernilai *active low*
- WR : berfungsi sebagai pengontrol kondisi status *write*, bernilai *active low*

- A0 – A1 : merupakan *bus address* yang berhubungan dengan RD dan WR, berfungsi sebagai *input* untuk menentukan *port* yang akan diakses
- PA0 – PA7: 8-bit *input* dan *output*.
- PB0 – PB7 : 8-bit *input* dan *output*.
- PC0 – PC7: 8-bit *input* dan *output*.

2.7.2. Diagram Blok

Diagram blok dari IC PPI 8255 ditunjukkan melalui gambar berikut ini:



Gambar 2.18. Arsitektur IC PPI 8255

Sumber: 82C55A *Datasheet*. (82C55A.*pdf*). San Jose: Harris Semiconductor, 1996. p.2.

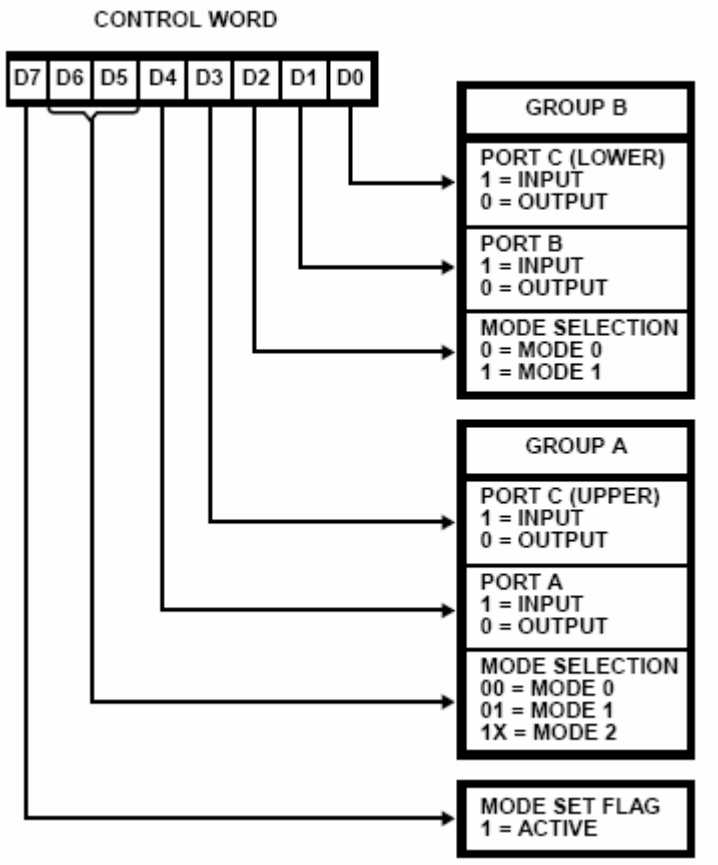
Data dari *microprocessor* masuk ke data *bus buffer* yang kemudian akan diteruskan ke *system* data *bus*. *Read Write Control Logic* akan mengolah *input* masukan dari *pin* RD, WR, A0, A1, *Reset* dan CS, *output*-nya akan mengontrol Grup A dan Grup B *Control. Control blocks* (Grup A dan Grup B) mengatur kondisi dari *Port*-A, *Port*-B dan *Port*-C. Konfigurasi *port* yang aktif dapat dilihat melalui tabel berikut ini :

Tabel 2.3. Konfigurasi *Port* 8255

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
1	1	0	1	0	Control Word → Data Bus
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					DISABLE FUNCTION
X	X	X	X	1	Data Bus → Three-State
X	X	1	1	0	Data Bus → Three-State

Sumber: 82C55A *Datasheet*. (82C55A.*pdf*). San Jose: Harris Semiconductor, 1996. p.3.

Sebelum digunakan, PPI 8255 harus diinisialisasi terlebih dahulu dengan cara memberikan data 8-bit yang disebut sebagai *control-word* ke dalam *Control Port* . Susunan *control-word* tersebut adalah sebagai berikut :



Gambar 2.19. *Control Word* 8255

Sumber: 82C55A *Datasheet*. (82C55A.*pdf*). San Jose: Harris Semiconductor, 1996. p.4.

Contoh : Jika menggunakan PPI dalam mode 0, dengan *Port*-A (PA4...PA7) sebagai *input*, *Port*-B (PB5...PB7) sebagai *output*, *Port*-C sebagai *output*, maka *control-word*nya adalah : $1\ 0\ 0\ 1\ 0\ 0\ 0\ 0_B = 90_H$