IMPLEMENTASI FPGA SEBAGAI SISTEM PEMONITORAN PADA PENGONTROL KECEPATAN MOTOR DC

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

RIZAL WAHYUDI

NIM. 115060301111005

KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2015

LEMBAR PERSETUJUAN

IMPLEMENTASI FPGA SEBAGAI SISTEM PEMONITORAN PADA PENGONTROL KECEPATAN MOTOR DC

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan

memperoleh gelar Sarjana Teknik

RIZAL WAHYUDI

NIM. 115060301111005

Skripsi ini telah direvisi dan disetujui oleh dosen pembimbing

pada tanggal 31 Juli 2015

Dosen Pembimbing I	**Dosen Pembimbing II**
Ir. Nanang Sulistiyanto, M.T.	**Mochammad Rif'an, S.T., M.T.**
NIP. 19700113 199403 1 002	**NIP. 19710301 200012 1 001**

LEMBAR PENGESAHAN

IMPLEMENTASI FPGA SEBAGAI SISTEM PEMONITORAN PADA PENGONTROL KECEPATAN MOTOR DC

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan

memperoleh gelar Sarjana Teknik

RIZAL WAHYUDI

NIM. 115060301111005

Skripsi ini telah diuji dan dinyatakan lulus sidang skripsi

pada tanggal 30 Juni 2015

Dosen Penguji I	**Dosen Penguji II**
Eka Maulana, S.T., M.T., M.Eng.	**Dr. Ir. Ponco Siwindarto, M.Eng.Sc.**
NIK. 841130 06 1 1 0280	**NIP. 19590304 198903 1 001**

Dosen Penguji III

Akhmad Zainuri, S.T., M.T.
NIP. 19840120 201212 1 003

Mengetahui,
Ketua Jurusan Teknik Elektro

M. Aziz Muslim, S.T., M.T., Ph.D.
NIP. 19741203 200012 1 001

PENGANTAR

Puji syukur kehadirat Allah SWT atas limpahan rahmat, taufiq dan hidayah-Nya sehingga penulisan skripsi dengan judul "IMPLEMENTASI FPGA SEBAGAI SISTEM PEMONITORAN PADA PENGONTROL KECEPATAN MOTOR DC" ini dapat terselesaikan dengan baik dan tepat pada waktunya. Shalawat serta salam penulis haturkan kepada junjungan kita Nabi Besar Muhammad SAW atas jasa beliau yang membawa kita dari zaman jahiliyah ke zaman yang terang benderang ini.

Penulis menyadari bahwa penulisan skripsi ini tidak akan terselesaikan tanpa adanya bantuan dari beberapa pihak. Pada kesempatan kali ini penulis menyampaikan terima kasih setulusnya kepada:

1. Ayahanda, Ibunda, dan adik tercinta yang senantiasa memberikan semangat, doa, kasih sayang, perhatian, serta dukungan baik materi maupun non-materi pada penulis selama ini.
2. Pemerintahan Direktorat Jenderal Pendidikan Tinggi (Dikti) yang telah memberikan beasiswa bidikmisi S1 untuk menuntut ilmu selama 4 tahun di Jurusan Teknik Elektro Universitas Brawijaya.
3. Bapak M. Aziz Muslim, ST., MT., Ph.D selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya dan Bapak Hadi Suyono, ST., MT., Ph.D selaku Sekertaris Jurusan Teknik Elektro Universitas Brawijaya.
4. Ibu Ir. Nurussa'adah, MT. selaku Ketua Kelompok Dosen Keahlian Elektronika Jurusan Teknik Elektro Universitas Brawijaya atas segala bimbingan pada ilmu elektronika, nasihat, pengarahan, motivasi, saran, dan masukan yang telah diberikan.
5. Bapak Ir. Nanang Sulistiyanto, M.T., selaku Dosen Pembimbing 1 atas segala bimbingan, nasihat, pengarahan, motivasi, saran, dan masukan yang telah diberikan, dan Bapak Mochammad Rif'an, S.T., M.T., selaku Dosen Pembimbing 2 atas segala bimbingan, nasihat, pengarahan, motivasi, saran, dan masukan yang telah diberikan.
6. Bapak dan Ibu dosen Jurusan Teknik Elektro Universitas Brawijaya yang telah memberikan dan mengajarkan ilmu elektro dengan setulus hati, baik secara langsung maupun tidak langsung pada perkuliahan.
7. Bapak dan Ibu staff dan karyawan Jurusan Teknik Elektro, baik secara langsung maupun tidak langsung telah membantu menyelesaikan skripsi ini.

8. Aprilianawati Sutarto dan Nurotul Auliya' teman seperjuangan dalam penelitian FPGA.
9. Keluarga besar Laboratorium Sistem Digital atas segala bantuan, pelajaran, dan dukungan yang telah diberikan.
10. Teman-teman Laboratorium Elektronika teman seperjuangan skripsi.
11. Teman-teman Konsentrasi Teknik Elektronika 2010, 2011 dan 2012 di Jurusan Teknik Elektro Universitas Brawijaya.
12. Teman-teman Inverter angkatan 2011 dan semua rekan-rekan di Jurusan Teknik Elektro Universitas Brawijaya Angkatan 2010 – 2013.
13. Seluruh teman-teman serta semua pihak yang tidak mungkin untuk dicantumkan namanya satu-persatu, terima kasih banyak atas segala bentuk bantuan dan dukungannya.

Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan. Maka dari itu, penulis mengharapkan kritik dan saran dari pembaca untuk membantu kesempurnaan skripsi ini. Akhir kata penulis berharap agar skirpsi ini dapat berguna dan bermanfaat bagi pembaca pada umumnya dan bagi mahasiswa Teknik Elektro Universitas Brawijaya pada khususnya.

Malang, Maret 2015

Penulis

DAFTAR ISI

halaman

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

RINGKASAN

Rizal Wahyudi, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Maret 2015, Implementasi FPGA Sebagai Sistem Pemonitoran Pada Pengontrol Kecepatan Motor DC, Dosen Pembimbing: Nanang Sulistiyanto dan Mochammad Rif'an.

FPGA menawarkan keuntungan proses aritmatika dan logika lebih cepat daripada mikrokontroler. Implementasi FPGA sebagai sistem pemonitoran diperlukan untuk memudahkan pengguna dalam menentukan nilai parameter PID khususnya pada sistem kontrol berbasis FPGA. Dalam penelitian ini, sistem pemonitoran diwujudkan dalam implementasi FPGA Xilinx Spartan 3E Nexys2 *kit board* menggunakan VHDL. Desain dan sintesis dari sistem pemonitoran digunakan *software* ISE *Design Suite* 14.6. Masukan sistem berupa *pin* PS/2 *Keyboard*, *pin* PWM dan *pin Rotary* sedangkan keluaran sistem berupa tampilan pemonitoran dari sinyal PWM dan sinyal *Rotary* serta nilai parameter PID pada VGA monitor. Masukan sinyal PWM dan sinyal *Rotary* serta nilai parameter PID dijadikan ke dalam bentuk data bitmap. Data-data bitmap diserempakan dengan *scan line* VGA monitor sebagai tampilan pixel dari sinyal PWM dan sinyal *Rotary* serta nilai parameter PID. Warna tampilan pixel pemonitoran pada VGA monitor hanya terdiri satu warna saja karena jumlah sumber daya memori internal yang terbatas pada Nexys2 *kit board*.

Pada hasil sintesis sistem pemonitoran, total sumber daya yang digunakan tidak melebihi kapasitas yang disediakan oleh Nexys2. Penggunaan sumber daya tersebut adalah *slice register* sebesar 1%, 4 *input* LUTs sebesar 4%, *occupied slice* sebesar 5%, *bonded* IOBs sebesar 10%, RAMB16s sebesar 40%, BUFGMUXs sebesar 8%, dan DCMs sebesar 25%. Berdasarkan hasil penelitian ini menunjukkan bahwa desain sistem pemonitoran dapat diimplementasikan pada FPGA Xilinx Spartan 3E Nexys2 *kit board* dengan penggunaan sumber daya terbesar adalah 40% atau sejumlah 8 dari 20 RAMB16s. Penggunaan sumber daya dibuat sedikit mungkin terutama pada proses aritmatika dan bisa menggunakan paket modul aritmatika yang disediakan oleh Nexys2 seperti DSP ataupun *Math Functions*. Dan sistem dapat diimplementasikan pada FPGA yang menyediakan sumber daya yang lebih banyak daripada Nexys2.

Kata kunci : VGA, VRAM, FPGA, Nexys2, sistem pemonitoran.

SUMMARY

Rizal Wahyudi, Department of Electrical Engineering, Faculty of Engineering, University of Brawijaya, March 2015, Implementation of FPGA as Monitoring System to DC Motor Speed Controller, Academic Supervisor : Nanang Sulistiyanto and Mochammad Rif'an.

FPGA offers the advantage of arithmetic and logical processes are faster than the microcontroller. FPGA implementation of a monitoring system is needed to facilitate the user in determining the PID parameter values, especially in FPGA-based control systems. In this study, monitoring system realized in the FPGA Xilinx Spartan 3E Nexys2 *kit board* implementation using VHDL. Design and synthesis of monitoring systems used ISE Design Suite 14.6 software. Input the system is PS/2 Keyboard pin, PWM pin and Rotary pin signal while output the system is monitoring display of PWM signal and Rotary signal and PID parameter values on the VGA monitor. Input of PWM signal and Rotary signal and PID parameter values changed into bitmap. Bitmap is synchronizing with the scan line VGA monitor as a display pixel of the PWM signal and the Rotary signal and PID parameter value. Color display pixel monitoring on VGA monitor consist only one color because the amount of internal memory resources limited on Nexys2 kit board.

On the results of the monitoring system synthesis, the total resources used doesn't exceed the capacity provided by Nexys2. Utilization of these resources is 1% slice register, 4% 4-input LUTs, 5% occupied slice, 10% bonded IOBs, 40% RAMB16s, 8% BUFGMUXs, and 25% DCMs. Based on the results of this study indicate that the design of monitoring system can be implemented at Nexys2 kit board Spartan 3E Xilinx FPGA with the use of resources is 40% or 8 by 20 RAMB16s. Thus Nexys2 FPGA can be used for DC Motor speed monitoring system. The use of resources made little as possible, especially in the process of arithmetic and it can use the arithmetic module package provided by Nexys2 like DSP or Math Functions. And the system can be implemented on FPGA resources that provides more than Nexys2.

Keywords : VGA, VRAM, FPGA, Nexys2, Monitoring system.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi perangkat keras (*hardware*) sudah terintegrasi dan penggunaannya yang mudah. Perangkat keras (*hardware*) tersebut dikemas ke dalam bentuk IC (*Integrated Circuit*) yang dapat diproses secara digital. FPGA (*Field Programmable Gate Array*) merupakan salah satu bentuk dari perangkat keras (*hardware*) digital tersebut yang memilki fungsi terpadu dan kemampuan proses sistem dengan kecepatan tinggi. FPGA sudah digunakan dalam skala besar pada pengontrolan sistem, ini dikarenakan FPGA memiliki aplikasi memproses sinyal digital secara cepat (DSP: Digital Signal Processing). Selain itu, FPGA mempunyai modul-modul logika yaitu logika kombinasional dan logika sekuensial yang bisa digunakan untuk proses sinyal secara kombinasional atau sekuensial. FPGA sebagai dasar pengontrolan menawarkan keuntungan yaitu sebagai komputasi yang cepat, fungsi-fungsi kompleks, kemampuan *real time processing*, dan konsumsi daya yang rendah. Lebar data yang fleksibel juga menjadi salah satu keuntungan lebih FPGA daripada mikroprosesor, mikrokontroler, dan *Digital Signal Processing* (DSP) sebab itu FPGA lebih baik untuk implementasi PID digital pada aplikasi generasi baru (Jogalekar,dkk, 2013).

Dalam bidang industri, Motor DC biasanya digunakan sebagai penggerak yang berbeban. Ketika Motor DC diberi beban maka kecepatan Motor DC akan turun, sehingga diperlukan pengontrol kecepatan Motor DC agar kecepatan Motor DC tetap sesuai dengan *setting point*. Para pengguna sistem kontrol untuk Motor DC sering kesulitan dalam menganalisis dan menentukan parameter sistem kontrol seperti nilai Kp, Ki, dan Kd tanpa melihat sinyal keluaran dari sistem kontrol Motor DC. Dalam kasus ini pengguna sistem kontrol Motor DC berbasis FPGA memerlukan sistem pemonitoran untuk melihat respon sinyal keluaran sistem kontrol. Dengan sistem pemonitoran ini akan memberikan efisiensi dalam penggunaan sistem kontrol berbasis FPGA dan kemudahan untuk menganalisis dan menentukan nilai parameter sistem kontrol Motor DC. Dalam kasus ini sistem pemonitoran yang diperlukan adalah sistem pemonitoran dengan proses aritmatika yang cepat, kecepatan proses sistem yang tinggi dan konsumsi daya yang rendah.

Dalam hal ini, sistem pemonitoran yang perlu dilakukan pada sistem kontrol Motor DC berbasis FPGA yaitu sistem yang memantau secara langsung terhadap sinyal

kontrol internal. Sinyal kontrol internal yang akan dipantau dalam sistem pemonitoran meliputi sinyal *Rotary* dan sinyal PWM serta nilai Kp, Ki, Kd, *setting point*. Dengan demikian untuk mewujudkan sistem pemonitoran dengan proses kecepatan dan kinerja yang tinggi serta konsumsi daya yang rendah tersebut maka digunakan perangkat keras (*hardware*) yaitu FPGA.

1.2 Rumusan Masalah

Dari permasalahan yang mengacu pada latar belakang, maka rumusan masalah dapat ditekankan sebagai berikut:

1. Seberapa cepatkah spesifikasi frekuensi VGA monitor yang digunakan dalam sistem pemonitoran?
2. Bagaimana implementasi FPGA sebagai sistem pemonitoran pada pengontrol kecepatan Motor DC?

1.3 Batasan Masalah

Dari permasalahan yang ada, maka pada sistematika pembahasan masalah diberi batasan-batasan sebagai berikut:

1. Sistem kontrol dalam proyek ini dilakukan pada penelitian lain.
2. *Driver* Motor dan sensor *Rotary* dalam proyek ini dilakukan pada penelitian lain.
3. Ketentuan nilai Kp, Ki, Kd, dan *setting point* dalam proyek ini dilakukan pada penelitian lain.
4. Komunikasi *Keyboard* yang digunakan dalam proyek ini adalah satu arah.
5. VGA Monitor yang digunakan dalam proyek ini dengan frekuensi 25 MHz.

1.4 Tujuan

Membuat sistem pemonitoran dengan kecepatan tinggi berbasis FPGA pada pengontrol kecepatan Motor DC.

1.5 Sistematika Penulisan

Bentuk sistematika penulisan dalam skripsi ini adalah:

BAB I Pendahuluan

Berisikan latar belakang, rumusan masalah, batasan masalah, tujuan dan sistematika penulisan.

BAB II Tinjauan Pustaka

Berisikan dasar teori yang mendukung dan menunjang dalam pembahasan yaitu mengenai FPGA (Look Up Table, Model Moore dan Model Mealy), *Keyboard* PS/2, VGA Monitor 640x480 pixel, dan VRAM.

BAB III Metode Penelitian

Berisikan tentang metode-metode yang akan dilakukan pada penelitian ini yaitu studi literatur (riset), spesifikasi (I/O, Proses, dan Memori), perancangan sistem pemonitoran, simulasi rangkaian, implementasi rangkaian, uji coba dan kesimpulan.

BAB IV Perancangan Sistem Pemonitoran

Berisikan tentang perancangan modul penyusun sistem pemonitoran pada pengontrol kecepatan Motor DC.

BAB V Hasil dan Pembahasan

Berisikan tentang hasil dan pembahasan mengenai simulasi dan pengujian pada modul penyusun sistem pemonitoran dan keseluruhan sistem pemonitoran.

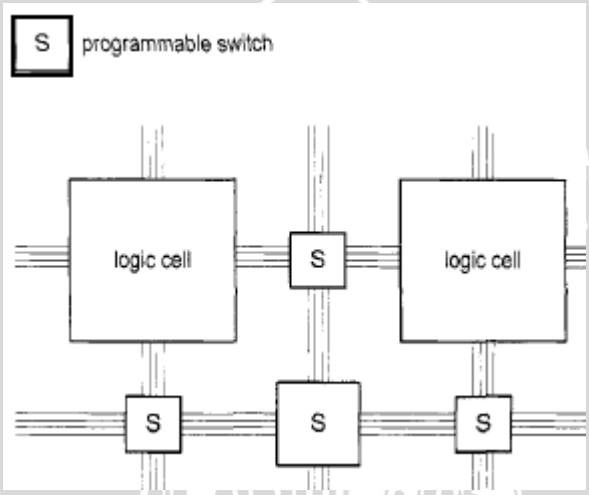
BAB VI Kesimpulan

Berisikan kesimpulan dan saran dari hasil pembahasan yang merujuk pada rumusan masalah dan tujuan.

BAB II
TINJAUAN PUSTAKA

2.1 FPGA (Field Programmable Gate Array)

FPGA (*Field Programmable Gate Array*) adalah suatu piranti keras yang tersusun atas array gerbang-gerbang logika dan *switchs* secara terpadu seperti Gambar 2.1. Gerbang logika dapat diprogram untuk membuat suatu fungsi rangkaian logika dan *switch* dapat dibuat sebagai penghubung jalur sinyal internal antara blok fungsi-fungsi rangkaian logika. FPGA merupakan tipe dari *integrated circuit* (IC) yang bisa diprogram untuk algoritma berbeda setelah pembuatan (Xilinx, 2013:11). Desain gerbang logika dan *switch* dibentuk ke dalam *chip* FPGA sehingga terbentuk rangkaian logika tersebut. Ketika proses ini selesai "di dalam bidang" daripada "fasilitas pembuatan (fab)," alat tersebut diketahui sebagai *field programmable* (Chu, 2008:12).

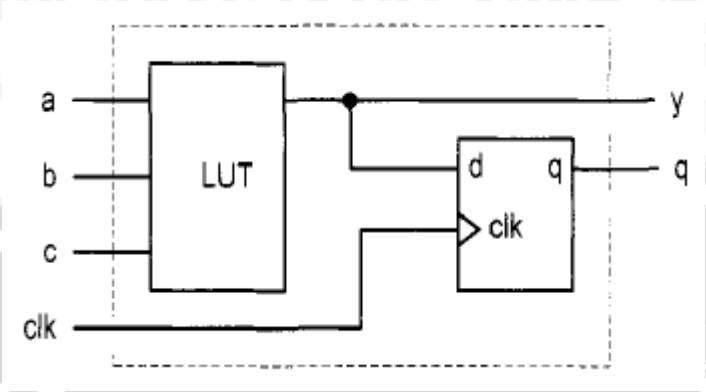


Gambar 2.1 Struktur *chip* FPGA.
Sumber: Chu (2008:12).

Sel-sel logika yang tesusun di dalam *chip* FPGA tersusun atas rangkaian-rangkaian kombinasional dan flip-flop D. Setiap susunan dari implementasi rangkaian kombinasional menggunakan metode LUT (*Look-Up Table*) seperti dalam Gambar 2.2. LUT adalah dasar yang membangun dari sebuah FPGA dan mampu mengimplementasikan fungsi logika dari *N* Boolean variabel (Xilinx, 2013:14). LUT tidak lepas dari tabel benaran sebuah kombinasi *n* masukan fungsi logika dengan nilai keluarannya. Sebuah *n* masukan LUT bisa dipertimbangkan sebagai memori kecil 2^nx1 *bit* (Chu, 2008:12).

FSM (*Finite State Machine*) adalah suatu model yang digunakan sebagai sistem aliran keadaan waktu transisi dari keadaan *internal*. Keadaan transisi ini dipengaruhi

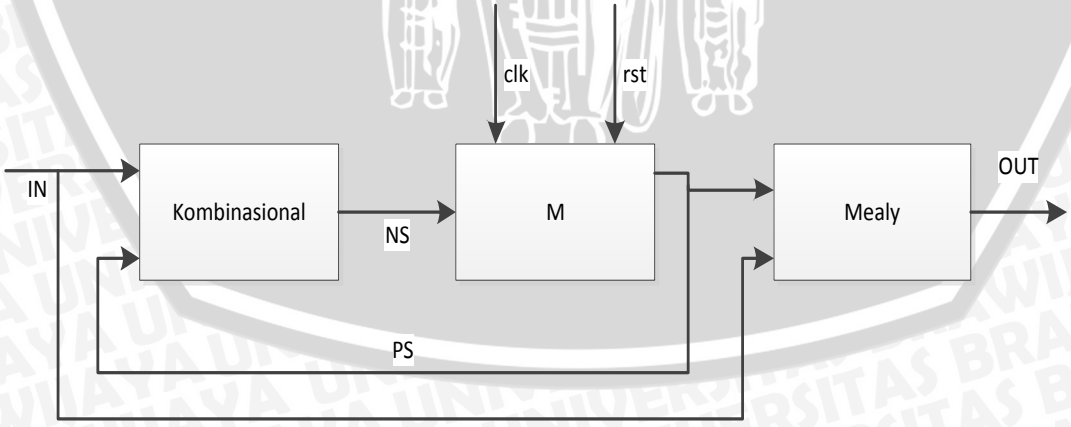
oleh masukan dan keadaan sekarang, pada FSM keadaan transisi tidak sederhana seperti pada rangkaian sekuensial (Chu, 2008:107). Pada rangkaian sekuensial logika keadaan selanjutnya itu sudah terstruktur sebagai penambah dan penggeser, sedangkan pada FSM logika keadaan selanjutnya itu terstruktur sebagai logika acak. Terdapat 2 jenis model FSM yaitu model Mealy dan Moore.



Gambar 2.2 3 masukan LUT.

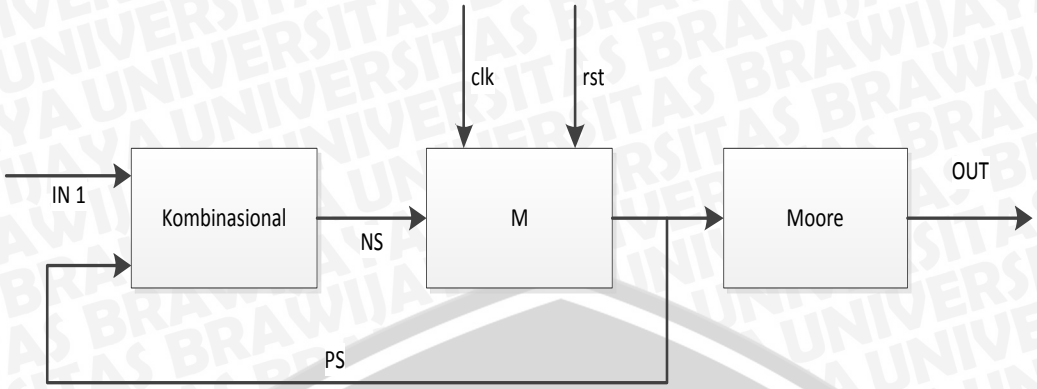
Sumber: Chu (2008:12).

Model Mealy dan model Moore sering digunakan pada bentuk FSM yang kompleks (Chu, 2008:107). Model Mealy adalah *output* fungsi FSM dipengaruhi oleh keadaan sekarang dan masukan. Model Moore adalah *output* fungsi FSM oleh keadaan sekarang saja. Perbedaan dari kedua model ini dapat dilihat pada fungsi keluarannya. Fungsi keluaran dari sistem Model Mealy dan Moore adalah sama tetapi tidak identik seperti dalam Gambar 2.3 dan Gambar 2.4.



Gambar 2.3 Model Mealy.

Sumber: Chu (2008:108).

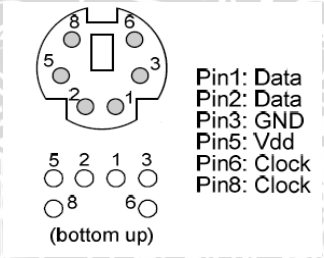


Gambar 2.4 Model Moore.

Sumber: Chu (2008:108).

2.2 Keyboard PS/2

PS/2 *port* seperti dalam Gambar 2.5 adalah *port* yang digunakan untuk komunikasi serial yang dikeluarkan oleh IBM personal komputer. PS/2 *port* memiliki 2 jalur untuk melakukan proses komunikasi data yaitu jalur data dan jalur *clock*. Jalur data digunakan sebagai jalur yang mengirimkan data secara serial. Jalur *clock* digunakan sebagai jalur pembawa informasi setiap periode *clock*. Jalur data tersebut memiliki paket data sebesar 11 *bit* yaitu 1 *bit* pertama sebagai *start bit*, 1 *bit* kedua sampai kesembilan sebagai data *bit*, 1 *bit* kesepuluh sebagai *parity bit* dan 1 *bit* kesebelas sebagai *stop bit*.



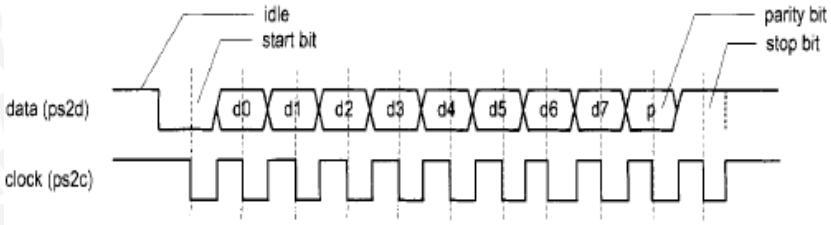
Gambar 2.5 PS/2 *Port*.

Sumber: Digilent (2011:7).

Komunikasi PS/2 *port* dengan *driver* bisa dilakukan secara dua arah atau satu arah. Komunikasi PS/2 *port* secara dua arah maka *driver* mengirimkan *command* kode ke *Keyboard* untuk menentukan parameter yang pasti. Komunikasi dua arah sangat sulit untuk diwujudkan pada *port Keyboard* PS/2, maka disarankan untuk menggunakan komunikasi satu arah dari *Keyboard* ke *driver board* (Chu, 2008:183).

Pada *timing diagram* PS/2 *port* seperti dalam Gambar 2.6, paket data yang dikirim oleh *Keyboard* dimulai dari *start bit*, 8 data *bit*, *parity bit* dan *stop bit*. Ketika terjadi *falling edge* pada sinyal PS/2 *clock* maka pada sinyal PS/2 data sudah valid dan

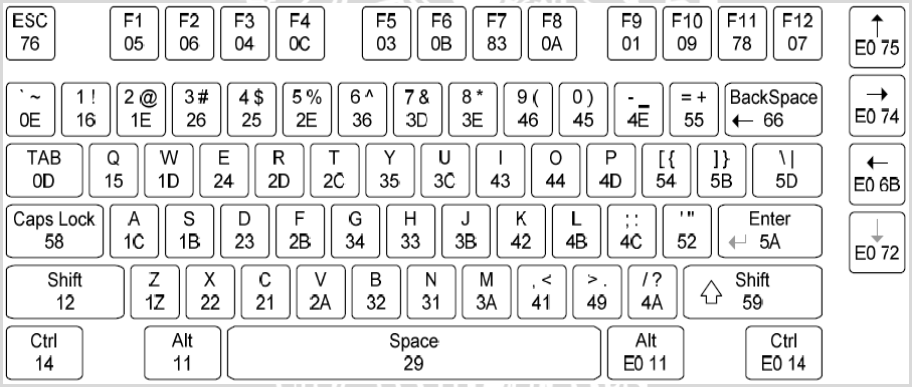
komunikasi baru bisa dilakukan kembali. Periode *clock* pada sinyal PS/2 *clock* yaitu antara 60 µs dan 100 µs (10 kHz sampai 16.7 kHz) dan sinyal PS/2 data akan stabil pada waktu sekurangnya 5 µs sebelum dan sesudahnya *falling edge* dari sinyal PS/2 *clock* (Chu, 2008:184).



Gambar 2.6 *Timing Diagram* PS/2 *port*.

Sumber: Chu (2008:184).

Keyboard PS/2 akan mengirimkan *make code* untuk komunikasi data ketika tombol ditekan pada *Keyboard* yang ditunjukkan dalam Gambar 2.7. Ketika tombol ditekan terus-menerus maka *make code* akan dikirim secara berulang-ulang setiap 100 ms. Ketika tombol pada *Keyboard* dilepaskan maka *break code* dikirimkan dari tombol tersebut.



Gambar 2.7 PS/2 *Keyboard* Standar IBM.

Sumber: Digilent (2011:9).

Make code terdiri dari 8 *bit* data atau 1 byte yang direpresentasikan sebagai 2 digit bilangan heksadesimal. Pada *Keyboard* PS/2, *make code* merupakan bagian utama dari data tombol *Keyboard* yang dikirim tersebut. Untuk tombol 0 maka *make code* tombol tersebut adalah 45. *Make code* ini bisa dikirim dalam satu paket data yang bisa terdiri dari 2 sampai 4 byte. Misalkan untuk tombol *upper arrow* maka *make code* tombol tersebut adalah E0 75, untuk *make code* tersebut dikirim secara *multiple* paket data. Ketika sesuatu tombol pada *Keyboard* dilepas maka *Keyboard* akan mengirimkan *break code* F0 setelah *make code*. Untuk tombol 0 maka *break code* tombol tersebut adalah F0

45. Jadi *Keyboard* akan mengirim *make code* kemudian *break code*, maka komunikasi data menjadi 45 F0 45. Dan jika tombol 0 ditekan secara terus-menerus sampai tombol dilepas, maka data yang dikirim oleh *Keyboard* adalah 45 45 45 45 45 ... 45 F0 45. Dalam fungsi *Keyboard*, *make code* diubah menjadi kode ASCII untuk merepresentasikan karakter Keyboard yang sesuai dengan *make code* itu sendiri. Kode ASCII adalah suatu bentuk kode yang merepresentasikan sebuah karakter atau *command*. Dalam *HyperTerminal* karakter yang dikirimkan terdiri dari 7 bit dan 128 kode yang mencakup huruf abjad, angka, dan simbol tanda baca (Chu, 2008:177). Kode ASCII ini ditunjukkan pada Tabel 2.1.

Tabel 2.1 Kode ASCII

Code	Char	Code	Char	Code	Char	Code	Char
00	(nul)	20	(sp)	40	@	60	`
01	(soh)	21	!	41	A	61	a
02	(stx)	22	"	42	B	62	b
03	(etx)	23	#	43	C	63	c
04	(eot)	24	$	44	D	64	d
05	(enq)	25	%	45	E	65	e
06	(ack)	26	&	46	F	66	f
07	(bel)	27	'	47	G	67	g
08	(bs)	28	(	48	H	68	h
09	(ht)	29	)	49	I	69	i
0a	(nl)	2a	*	4a	J	6a	j
0b	(vt)	2b	+	4b	K	6b	k
0c	(np)	2c	,	4c	L	6c	l
0d	(cr)	2d	-	4d	M	6d	m
0e	(so)	2e	.	4e	N	6e	n
0f	(si)	2f	/	4f	O	6f	o
10	(dle)	30	0	50	P	70	p
11	(dc1)	31	1	51	Q	71	q
12	(dc2)	32	2	52	R	72	r
13	(dc3)	33	3	53	S	73	s
14	(dc4)	34	4	54	T	74	t
15	(nak)	35	5	55	U	75	u
16	(syn)	36	6	56	V	76	v
17	(etb)	37	7	57	W	77	w
18	(can)	38	8	58	X	78	x
19	(em)	39	9	59	Y	79	y
1a	(sub)	3a	:	5a	Z	7a	z
1b	(esc)	3b	;	5b	[	7b	{
1c	(fs)	3c	<	5c	\	7c	|
1d	(gs)	3d	=	5d	]	7d	}
1e	(rs)	3e	>	5e	^	7e	~
1f	(us)	3f	?	5f	_	7f	(del)

Sumber: Chu (2008:179)

2.3 VGA Monitor

VGA (Video Graphics Array) adalah suatu standar *display* dari IBM personal komputer dengan dasar warna yang terdiri dari 256 jenis warna. VGA VGA monitor menggunakan VGA *port* untuk berkomunikasi dengan *driver* ataupun pengontrol VGA. VGA *port* terdiri dari 8 *bit* warna *display* dengan warna yang utama terdiri dari warna merah, hijau dan biru seperti dalam Gambar 2.8. VGA *port* juga terdiri dari 2 sinyal sinkron yang mencakup sinyal horizontal sinkron dan vertikal sinkron.

Red (R)	Green (G)	Blue (B)	Resulting color
0	0	0	black
0	0	1	blue
0	1	0	green
0	1	1	cyan
1	0	0	red
1	0	1	magenta
1	1	0	yellow
1	1	1	white

Gambar 2.8 Kombinasi 3 *Bit* Warna.

Sumber: Chu (2008:259).

Sinyal horizontal sinkron digunakan sebagai *scan* kolom pada *display* VGA yang ditunjukkan dalam Gambar 2.9 dan sedangkan sinyal vertikal sinkron digunakan sebagai *scan* baris pada *display* VGA yang ditunjukkan dalam Gambar 2.10. VGA monitor 640x480 pixel mempunyai daerah *blank* yaitu untuk kolom 160 pixel dan baris 45 pixel. Maka ukuran total VGA monitor 640x480 pixel adalah 800 pixel 525 baris. Untuk ukuran *display* VGA monitor 640 pixel 480 baris harus di*scan* dengan frekuensi sebesar 25 MHz, artinya dalam pixel monitor akan di*scan* sebanyak 25 Mpixel dalam 1 detik sesuai dengan Persamaan (2-4). Koordinat pixel pada *display* VGA monitor 640 pixel 480 baris dari kiri atas sampai kanan bawah adalah (0,0) sampai (639,479). Untuk mendapatkan frekuensi *scan* pixel dengan resolusi monitor 640x480 pixel dapat ditentukan dengan cara:

$$p = 800\ \frac{pixel}{line} \tag{2-1}$$

$$l = 525\ \frac{line}{layar} \tag{2-2}$$

$$s = 60\ \frac{layar}{detik} \tag{2-3}$$

$$frekuensi\ scan\ pixel = p * l * s\ \approx 25\ M\ \frac{pixel}{detik} \tag{2-4}$$

Keterangan:

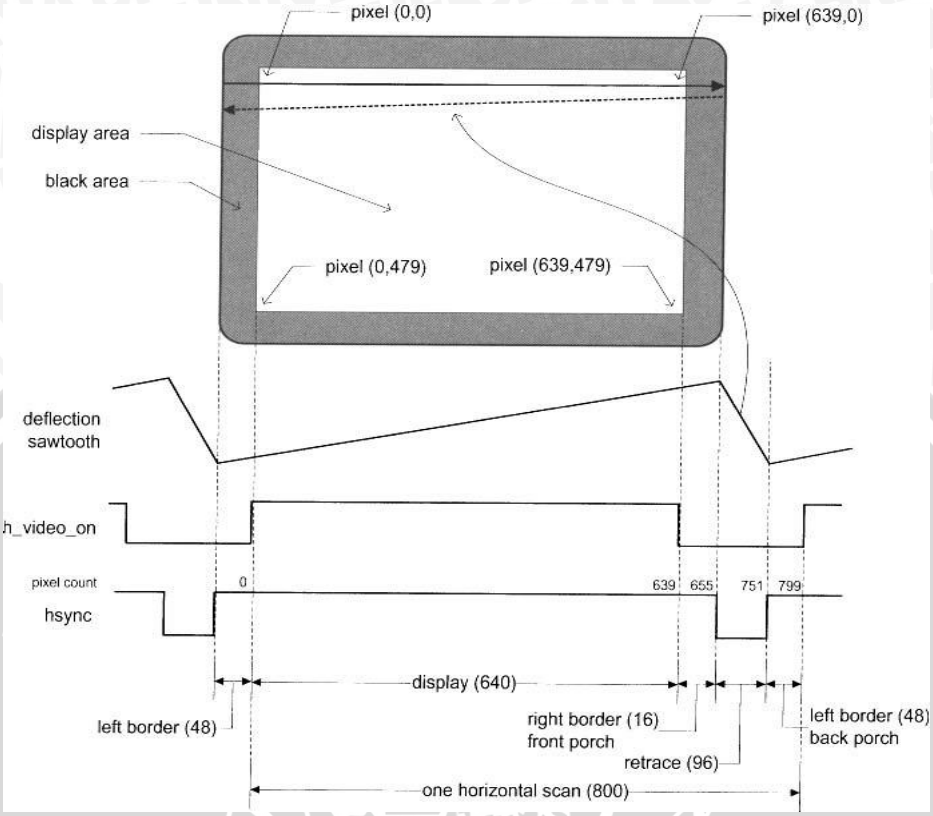
p = jumlah pixel per baris

l = jumlah baris per layar

s = jumlah *scan* layar per detik

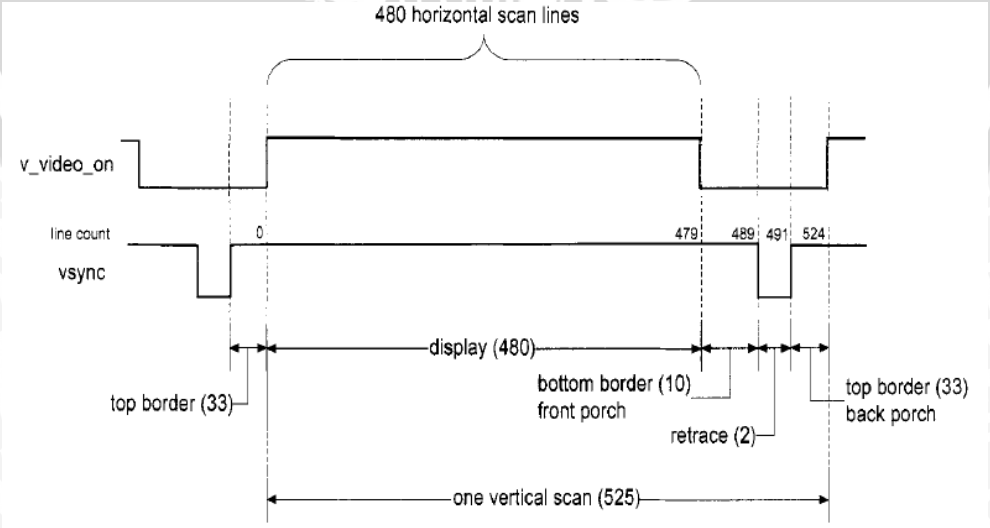
Pada sinyal horizontal *scan*, *display* tersebut merupakan daerah *visible* dimana pixel yang bisa ditampilkan pada layar dengan panjang daerah 640 pixel. *Retrace* adalah daerah dimana titik elektron kembali pada posisi awal kiri layar, maka keadaan sinyal video akan gelap dengan panjang daerah sebesar 96 pixel. *Right border* merupakan daerah batas *visible display* kanan (*front porch*), dimana pada daerah ini sinyal video

akan gelap dengan panjang daerah sebesar 16 pixel. *Left border* merupakan daerah batas *visible display* kiri (*back porch*), dimana pada daerah ini sinyal video akan gelap dengan panjang daerah sebesar 48 pixel.



Gambar 2.9 *Timing Diagram* Sinyal Horizontal *Scan*.

Sumber: Chu (2008:261).



Gambar 2.10 *Timing Diagram* Sinyal Vertikal *Scan*.

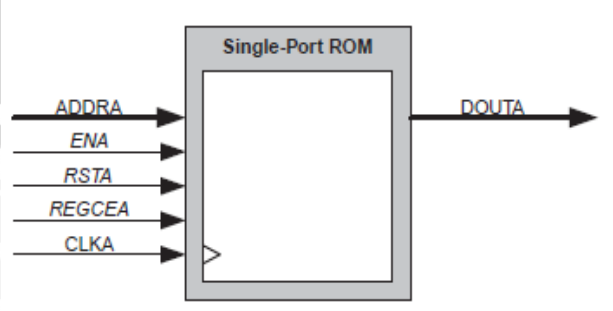
Sumber: Chu (2008:262).

Pada sinyal vertikal *scan*, *display* tersebut merupakan daerah *visible* dimana pixel yang bisa ditampilkan pada layar dengan panjang daerah 480 *line*. *Retrace* adalah daerah dimana titik elektron kembali pada posisi awal atas layar, maka keadaan sinyal video akan gelap dengan panjang daerah sebesar 2 *line*. *Bottom border* merupakan daerah batas *visible display* bawah (*front porch*), dimana pada daerah ini sinyal video akan gelap dengan panjang daerah sebesar 10 *line*. *Top border* merupakan daerah batas *visible display* atas (*back porch*), dimana pada daerah ini sinyal video akan gelap dengan panjang daerah sebesar 33 *line*.

2.4 VRAM

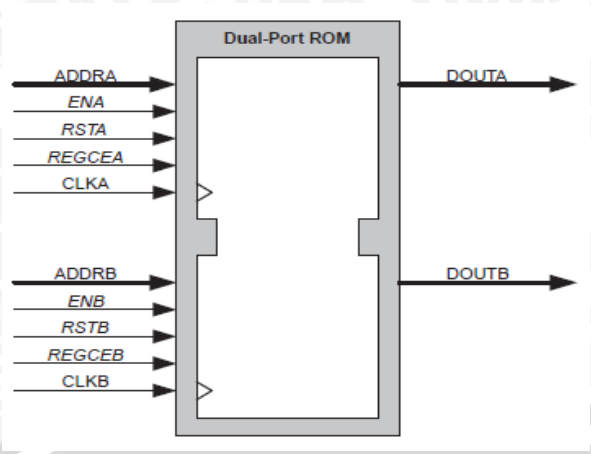
VRAM (Video RAM) adalah RAM yang digunakan sebagai tempat menyimpan bitmap gambar atau data gambar yang akan ditampilkan pada layar monitor. Koordinat *horzontal scan* dan *vertikal scan* pada *display* digunakan sebagai pengalamatan RAM untuk data gambar. VRAM dibangun oleh blok memori RAM yang disediakan pada FPGA. Tipe blok memori yang disediakan oleh FPGA ada 5 tipe yaitu *single port* RAM, *simple dual port* RAM, *true dual port* RAM, *single port* ROM dan *dual port* ROM.

Single port ROM yang ditunjukkan dalam Gambar 2.11 menyediakan akses baca ke alokasi memori melalui *single port*. *Dual port* ROM seperti dalam Gambar 2.12 menyediakan akses baca ke alokasi memori melalui dua *port*. *Single port* RAM seperti dalam Gambar 2.13 menyediakan akses baca dan tulis ke alokasi memori melalui *single port*. *Simple dual port* RAM seperti dalam Gambar 2.14 menyediakan 2 *port* A dan B, dengan ketentuan *port* A sebagai akses tulis ke alokasi memori dan *port* B sebagai akses baca ke alokasi memori. *True dual port* RAM dalam Gambar 2.15 menyediakan 2 *port* A dan B, dengan ketentuan akses baca dan tulis ke alokasi memori bisa dilakukan melalui salah satu dari *port*.



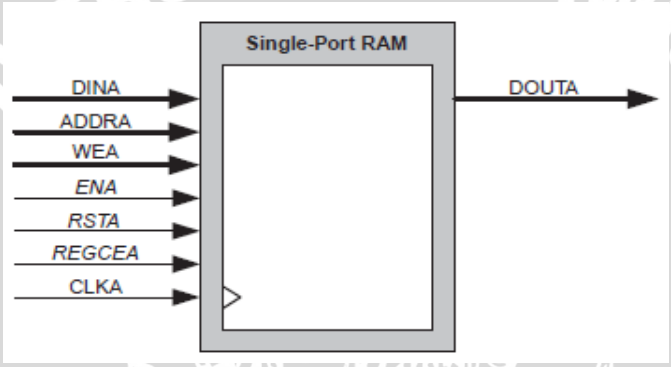
Gambar 2.11 *Single port* ROM.

Sumber: Xilinx (2011:19).



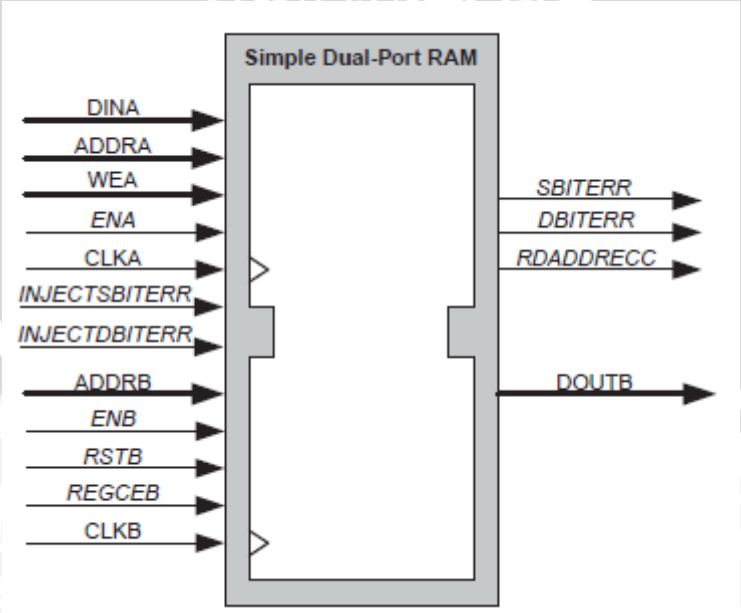
Gambar 2.12 *Dual port* ROM.

Sumber: Xilinx (2011:20).



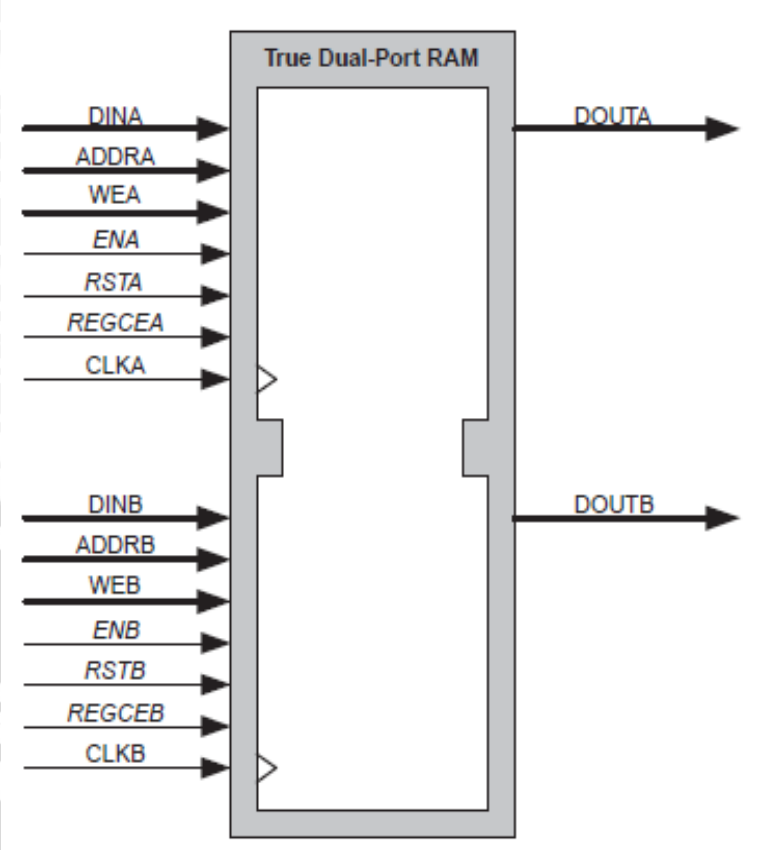
Gambar 2.13 *Single port* RAM.

Sumber: Xilinx (2011:20).



Gambar 2.14 *Simple dual port* RAM.

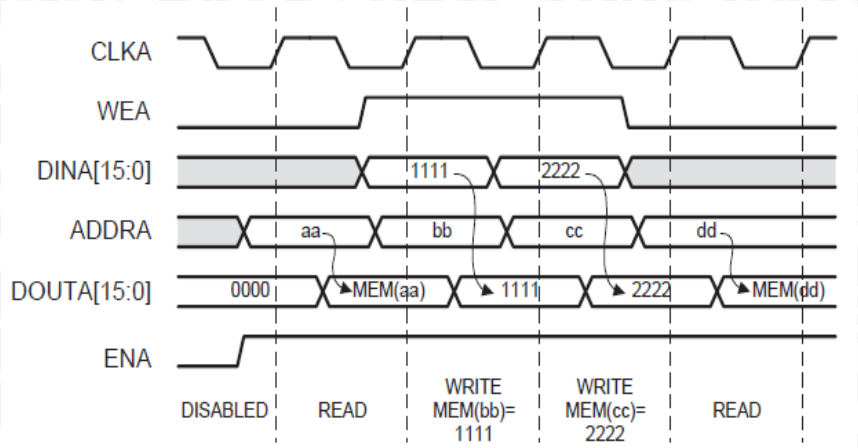
Sumber: Xilinx (2011:21).



Gambar 2.15 *True dual port* RAM.

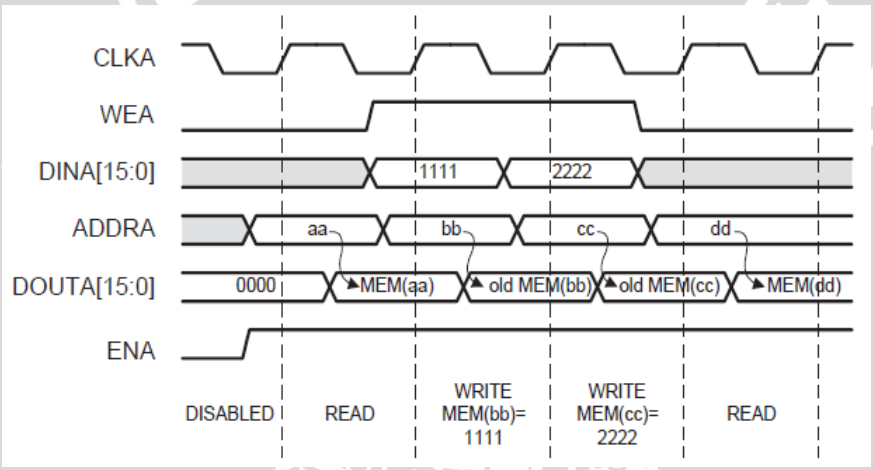
Sumber: Xilinx (2011:21).

Mode operasi dalam blok memori generator untuk setiap *port* ditentukan oleh hubungan antara antarmuka *port* baca dan tulis (Xilinx, 2011:25). Terdapat tiga jenis *mode* operasi dalam blok memori generator. Tiga jenis *mode* operasi ini digunakan untuk mengoperasikan *port* A maupun *port* B pada blok memori generator. Tiga jenis *mode* operasi tersebut ialah *write first mode*, *read first mode* dan *no change mode*. Jika terdapat pengalamatan yang bentrok terhadap *port* A dan *port* B maka satu dari tiga jenis *mode* operasi ini mempunyai efek terhadap hubungan antara *port* A dan *port* B. *Mode* operasi *Write first mode* yang ditunjukkan dalam Gambar 2.16 merupakan *mode* yang mengutamakan data *input* ditulis secara serempak ke dalam memori sehingga data *output* segera terisi sesuai dengan data *input* selama proses operasi *write*. Read *first mode* seperti dalam Gambar 2.17 merupakan *mode* yang mengutamakan data *output* dibaca sebelumnya pada memori sesuai dengan alamat yang ditulis pada data *output* tersebut sementara data *input* disimpan pada memori. *No change mode* dalam Gambar 2.18 merupakan *mode* dengan data *output* tertahan tidak terjadi perubahan pada bus data *output* ketika selama proses operasi *write*, tetapi data *output* tetap dibaca sebelumnya dan tidak dipengaruhi oleh proses operasi *write* pada *port* yang sama tersebut.



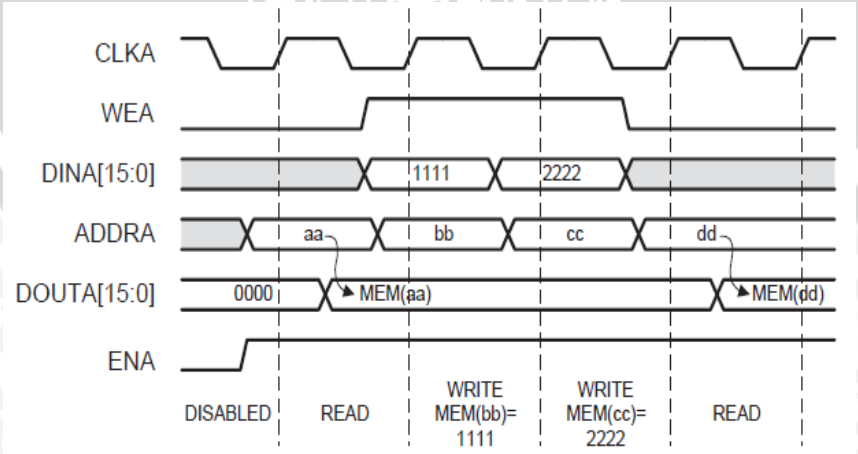
Gambar 2.16 *Timing Diagram Write First Mode Operation*.

Sumber: Xilinx (2011:26).



Gambar 2.17 *Timing Diagram Read Mode Operation*.

Sumber: Xilinx (2011:26).



Gambar 2.18 *Timing Diagram No Change Mode Operation*.

Sumber: Xilinx (2011:27).

BAB III
METODE PENELITIAN

Metode penelitian dilakukan untuk menyelesaikan rumusan masalah dalam penyusunan skripsi ini. Metode penelitian dalam skripsi ini mencakup studi literatur (riset), penentuan spesifikasi (I/O, Proses, dan Memori), perancangan *hardware*, simulasi rangkaian, implementasi rangkaian, dan uji coba. Langkah-langkah tersebut dilakukan secara bertahap dan teratur agar penyelesaian laporan penelitian ini tepat pada waktunya.

3.1 Studi Literatur

Studi literatur ini dilakukan selama satu tahun secara khusus mengenai penelitian atau riset implemnetasi FPGA sebagai pemonitoran pengontrol kecepatan motor DC. Secara umum dasar teori penunjang untuk studi literatur ini didapat dari pelajaran studi kuliah yaitu bahasa diskripsi perangkat keras, elektronika digital, arsitektur komputer, dan teknik digital. Studi literatur ini berkaitan dengan bahasan dasar FPGA, bahasa VHDL, Look Up Table, memori, dan implementasi logika.

3.2 Spesifikasi Rancangan Implementasi

1. Spesifikasi *input/output* rancangan implementasi

   Spesifikasi *input/output* untuk modul FPGA adalah PS/2 *port* standar IBM, VGA *port* standar IBM. Modul FPGA yang digunakan adalah keluarga Xilinx Spartan-3E Nexys 2. Spesifikasi modul FPGA ini antara lain adalah 648 Kb blok RAM, 500K gerbang logika, dan frekuensi *clock* eksternal 50 MHz.

2. Spesifikasi proses rancangan implementasi

   Spesifikasi untuk proses *input* dalam rancangan implementasi ini adalah berupa sinyal PWM dengan frekuensi 1 KHz, sinyal *Rotary* dan *Keyboard* PS/2 standar IBM sebagai masukan/karakter untuk nilai Kp, Ki, Kd dan *setting point*. Karakter *Keyboard* yang hanya digunakan yaitu ‘0’, ’1’, ’2’, ’3’, ’4’, ’5’, ’6’, ’7’, ’8’, ’9’, *enter*, *backspace*, dan *command* berupa *arrow upper* dan *arrow down*. Dan spesifikasi untuk proses *output* dalam rancangan implementasi ini adalah VGA monitor 640x480 pixel standar IBM dengan frekuensi *scan* pixel 25 MHz sebagai tampilan pemonitoran sinyal PWM, sinyal *Rotary*, nilai Kp, Ki, Kd, dan *setting point*. Tempat tampilan VGA monitor dibagi mejadi tiga bagian yaitu tempat tampilan nilai Kp, Ki, Kd, dan *setting point* 48x64 pixel, tempat tampilan sinyal PWM 480x64 pixel, sinyal *Rotary* 480x64 pixel.

3. Spesifikasi memori rancangan implementasi

Spesifikasi memori yang digunakan dalam perancangan *Keyboard* untuk setiap karakter ASCII 256x8 *bit* atau 256 *Byte*. Spesifikasi memori yang digunakan dalam perancangan VGA untuk setiap ukuran karakter 8x16 pixel sebanyak 4 baris, tampilan sinyal PWM 480x64 pixel dan tampilan sinyal *Rotary* 480x64 pixel. *Space* memori yang dibutuhkan untuk tampilan karakter sebesar 48x64x1 *bit* atau 3 Kb, *space* memori yang dibutuhkan untuk tampilan sinyal PWM sebesar 480x64x1 *bit* atau 32 Kb dan untuk tampilan sinyal *Rotary* sebesar 480x64x1 *bit* atau 32 Kb. Jadi total keseluruhan *space* memori yang dibutuhkan sebesar 67 Kb.

3.3 Perancangan Sistem Pemonitoran

Penyusunan arsitektur sistem pemonitoran dilakukan sebelum membuat rancangan modul sistem pemonitoran pada modul FPGA. Modul sistem pemonitoran yang kompleks dirancang berdasarkan per bagian sistem. Perancangan per bagian sistem dari sistem pemonitoran dilakukan untuk mempermudah dalam memprogram *hardware* sistem pemonitoran menggunakan VHDL dari *software* ISE *Design Suite* 14.6. Bagian sistem dari sistem pemonitoran tersebut mencakup perancangan *Keyboard driver*, *Keyboard converter*, *scan code*, *scan* PWM dan *Rotary*, VRAM dan VGA *driver*.

3.4 Simulasi Rangkaian

Simulasi rangkaian dilakukan pada setiap blok proses rangkaian sistem yaitu simulasi *Keyboard driver*, *Keyboard converter*, *scan code*, *scan* PWM dan *Rotary*, VRAM, VGA *driver*. Dan simulasi rangkaian secara menyeluruh agar blok rangkaian dapat berkerja dengan baik dan sesuai dengan proses setiap blok. Parameter yang perlu diperhatikan dalam simulasi adalah pada *port* keluaran yang disesuaikan dengan masukan *port*. *Timing* proses simulasi sangat penting untuk melihat waktu perubahan keluaran sistem. Aplikasi simulasi yang digunakan adalah ISIM *Simulation* dari FPGA Xilinx.

3.5 Implementasi Rangkaian

Implementasi rangkaian dilakukan pada rangkaian *Keyboard* PS/2 dan monitor VGA 640x480 pixel. Implementasi dilakukan untuk melihat fungsi kinerja rangkaian. Hasil dari kinerja rangkaian disesuaikan dengan hasil simulasi yang kemudian akan dilakukan pengambilan data pada uji coba rangkaian.

3.6 Uji Coba Rangkaian

Uji coba rangkaian dilakukan pada setiap blok rangkaian dan keseluruhan blok rangkaian. Uji coba ini dilakukan untuk melihat data keluaran setiap blok rangkaian dan keseluruhan blok rangkaian. Hasil uji coba disesuaikan dengan fungsi kinerja dan hasil simulasi proses rangkaian. Pengujian blok dilakukan secara bertahap dan berurutan setiap blok rangkaian yaitu:

1. Uji coba rangkaian *Keyboard driver*

   Uji coba yang dilakukan pada rangkaian *Keyboard driver* bertujuan untuk melihat *make code* untuk setiap karakter atau *command* pada *Keyboard*. Dengan menggunakan *Keyboard* PS/2 sebagai *input* rangkaian dan 8 *bit* led sebagai *output* rangkaian. Dengan menekan tombol *Keyboard* dan melihat keluaran 8 *bit* LED sebagai bentuk dari *make code*, apakah keluaran 8 *bit* LED sesuai dengan sandi *make code* tombol *Keyboard* tersebut.

2. Uji coba rangkaian *Keyboard converter*

   Uji coba yang dilakukan pada rangkaian *Keyboard converter* bertujuan untuk melihat kode ASCII untuk setiap sandi *make code*. Dengan menggunakan 8 *bit switch* sebagai *input* rangkaian dan 8 *bit* LED sebagai *output* rangkaian. Dengan kombinasi 8 *bit switch* sebagai representasi *make code* dan melihat keluaran 8 *bit* LED sebagai bentuk dari sandi ASCII, apakah keluaran 8 *bit* LED sesuai dengan sandi ASCII dari *make code* tersebut.

3. Uji coba rangkaian *scan code*

   Uji coba yang dilakukan pada rangkaian *scan code* bertujuan untuk melihat karakter untuk setiap sandi ASCII. Dengan menggunakan 8 *bit switch* sebagai *input* alamat ROM, 8 *bit* LED sebagai *output* data ROM. Dengan kombinasi 8 *bit switch* sebagai representasi alamat ROM dan melihat keluaran 8 *bit* LED sebagai bentuk dari data ROM, apakah keluaran 8 *bit* LED sesuai dengan data di ROM pada alamat tersebut.

4. Uji coba rangkaian VRAM

   Uji coba yang dilakukan pada rangkaian VRAM bertujuan untuk melihat data *bit* pixel tersimpan pada VRAM. Dengan menggunakan 1 *bit button* sebagai *input* tombol *write*, 7 *bit switch* sebagai *input* alamat memori, 1 *bit switch* sebagai *input* 1 *bit* data dan LED 1 *bit* sebagai tampilan data *bit*. Dengan kombinasi 7 *bit switch* sebagai representasi alamat memori dan menentukan 1 *bit switch* data *bit* kemudian dengan menekan tombol 1 *bit button write* dan melihat

tampilan data *bit* LED, apakah data *bit* tersimpan sesuai dengan data *bit* pada kombinasi alamat 7 *bit switch*.

5. Uji coba rangkaian VGA *driver*

   Uji coba yang dilakukan pada rangkaian VGA *drive* bertujuan untuk melihat area VGA *display*. Dengan menggunakan 8 *bit switch* sebagai *input* warna tampilan VGA. Dengan kombinasi 8 *bit switch* sebagai representasi kombinasi RGB dan melihat tampilan warna pada monitor VGA, apakah tampilan sesuai dengan warna pada kombinasi 8 *bit switch* tersebut.

6. Uji coba rangkaian keseluruhan sistem pemonitoran

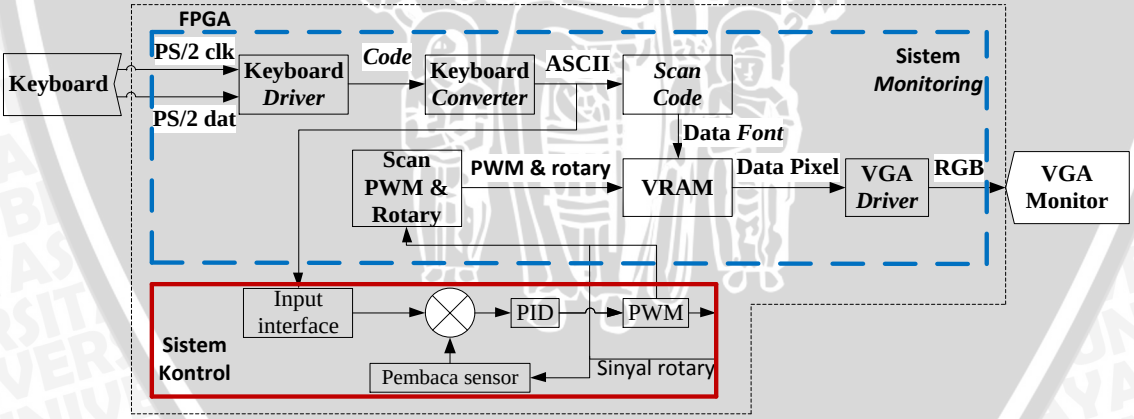
   Uji coba yang dilakukan pada rangkaian keseluruhan sistem pemonitoran bertujuan untuk melihat karakter *Keyboard* dapat ditampilkan pada VGA monitor. Dengan menggunakan *Keyboard* PS/2 sebagai masukan sistem pemonitoran. Dan dengan menggunakan VGA monitor sebagai keluaran sistem pemonitoran, apakah tampilan karakter pada VGA monitor sesuai dengan masukan tombol *Keyboard*.

BAB IV
PERANCANGAN SISTEM PEMONITORAN

4.1 Arsitektur Sistem

Perancangan arsitektur sistem seperti dalam Gambar 4.1 dalam blok garis putus-putus dilakukan sebelum merancang sistem pemonitoran. Dalam perancangan sistem pemonitoran dibutuhkan dua *port* pada FPGA yaitu *Keyboard port* dan VGA *port*. Kerja sistem dimulai dari blok *Keyboard driver*. Blok *Keyboard driver* membaca masukan tombol *Keyboard* berupa *make code*. Kemudian *make code* diterjemahkan ke dalam bentuk kode ASCII pada blok *Keyboard converter*. Masukan kode ASCII digunakan untuk membaca karakter ROM pada blok *scan code*. Hasil pembacaan data *bit* pada ROM tersebut berupa *data font*. Keluaran sinyal PWM dan sensor *Rotary* berupa data *bit* merupakan hasil proses dari proyek lain yang kemudian diproses pada blok *scan* PWM dan *Rotary*. Keluaran data *bit* PWM dan *Rotary* serta data *font* tersebut diproses dan disimpan pada blok VRAM.

Data *bit* yang tersimpan pada memori blok VRAM kemudian dibaca dengan keluaran berupa data pixel. Data pixel diproses pada blok VGA *driver* dengan keluaran sebagai bentuk RGB. Kemudian Keluaran RGB tersebut ditampilkan pada VGA monitor.



Gambar 4.1 Diagram blok keseluruhan sistem.

Keterangan:

Proses dalam proyek pada penelitian ini

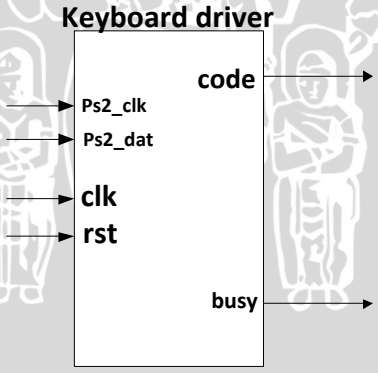
Proses dalam proyek pada penelitian lain

4.2 Perancangan Sistem Pemonitoran

Sistem pemonitoran dirancang berdasarkan arsitektur sistem pemonitoran. Perancangan sistem dilakukan secara urut dan disusun sesuai dengan arsitektur sistem. Perancangan sistem pemonitoran dimulai dari perancangan modul *Keyboard driver*, modul *Keyboard converter*, modul *scan code*, modul *scan* PWM dan *Rotary*, modul VRAM, modul VGA *driver*, dan modul Gen_clk.

4.2.1 Perancangan Sistem Modul *Keyboard Driver*

Sistem modul *Keyboard driver* seperti dalam Gambar 4.2 dirancang berdasarkan *timing diagram* PS/2 *port* dalam Gambar 2.6. Operasi modul *Keyboard driver* ialah mengolah data *bit* tombol *Keyboard* menjadi *make code*. Berdasarkan prinsip kerja *Keyboard driver* tersebut maka perancangan sistem modul *Keyboard driver* menggunakan proses sekuensial dengan model Moore. Jika tombol *Keyboard* ditekan maka *Keyboard* akan mengirimkan sinyal *clock* pada ps2_clk dan sinyal data pada ps2_dat. Setiap terjadinya *falling edge* ps2_clk maka *counter* atau *cnt* akan bertambah satu untuk menghitung jumlah ps2_clk dan *register* geser sinkron 11 *bit* akan tergeser ke kanan 1 *bit* dengan data ps2_dat. Jika tidak terjadinya *falling edge* pada ps2_clk maka nilai *counter* sama dengan nilai *counter* sebelumnya dan data *register* ps2_dat sama dengan data sebelumnya.

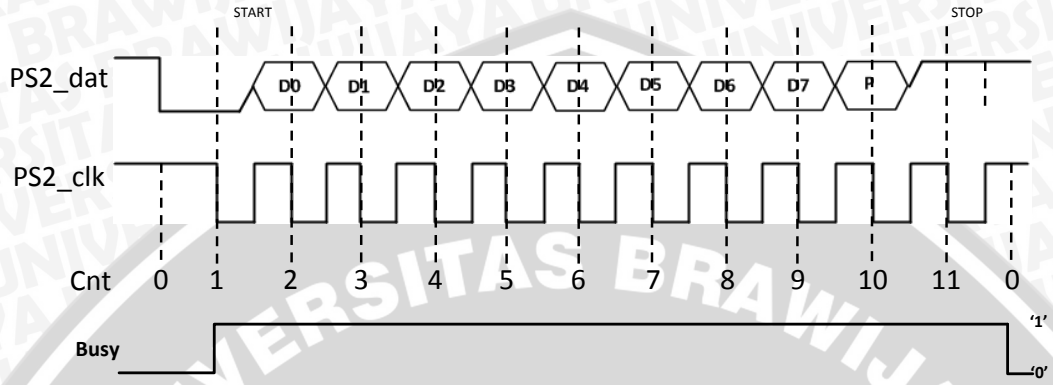


Gambar 4.2 Modul *Keyboard driver*.

Keterangan:

- clk = masukan sumber sinyal *clock* 25 MHz
- rst = masukan sinyal reset
- ps2_clk = masukan *port* PS/2 *clock*
- ps2_dat = masukan *port* PS/2 data
- *code* = keluaran sinyal *code* 8 *bit*
- *busy* = keluaran sinyal proses baca *Keyboard*

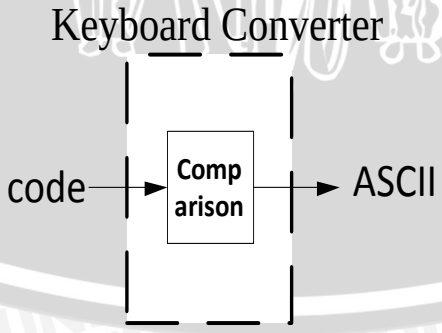
Proses operasi dari modul *Keyboard driver* dapat ditunjukkan dalam Gambar 4.3. Jika jumlah *cnt* > 0 dan *cnt* <= 11 maka *busy* akan '1' sedangkan jika tidak maka *busy* akan '0'. Jika jumlah *cnt* = 11 dan keadaan ps2_clk '1' maka nilai *cnt* kembali menjadi 0. Hasil keluaran *code* diambil pada 8 *bit register* ps2_dat mulai dari *bit* ke-1 sampai *bit* ke-8 sebagai *make code*, *command code* atau *break code*.



Gambar 4.3 *Timing diagram* perancangan *Keyboard driver*.

4.2.2 Perancangan Sistem Modul *Keyboard Converter*

Sistem modul *Keyboard converter* seperti dalam Gambar 4.4 dirancang berdasarkan tabel kode ASCII dalam Tabel 2.1 Kode ASCII. Operasi modul *Keyboard converter* yaitu membandingkan masukan *code Keyboard* dengan *make code*, *break code* dan *command code*. Jika *code Keyboard* berupa *make code* maka *code* akan langsung diterjemahkan ke dalam bentuk kode ASCII. Dan jika *code Keyboard* berupa *break code* atau berupa *command code* maka *code Keyboard* tersebut diabaikan sampai *code* berupa *make code*.



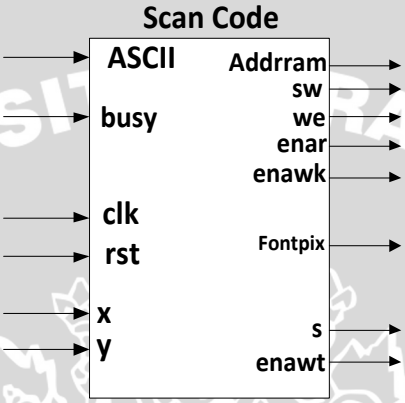
Gambar 4.4 Modul *Keyboard converter*.

Keterangan:

- *code* = masukan sinyal *code* 8 *bit*
- ascii = keluaran sinyal kode ASCII 8 *bit*

4.2.3 Perancangan Sistem Modul *Scan Code*

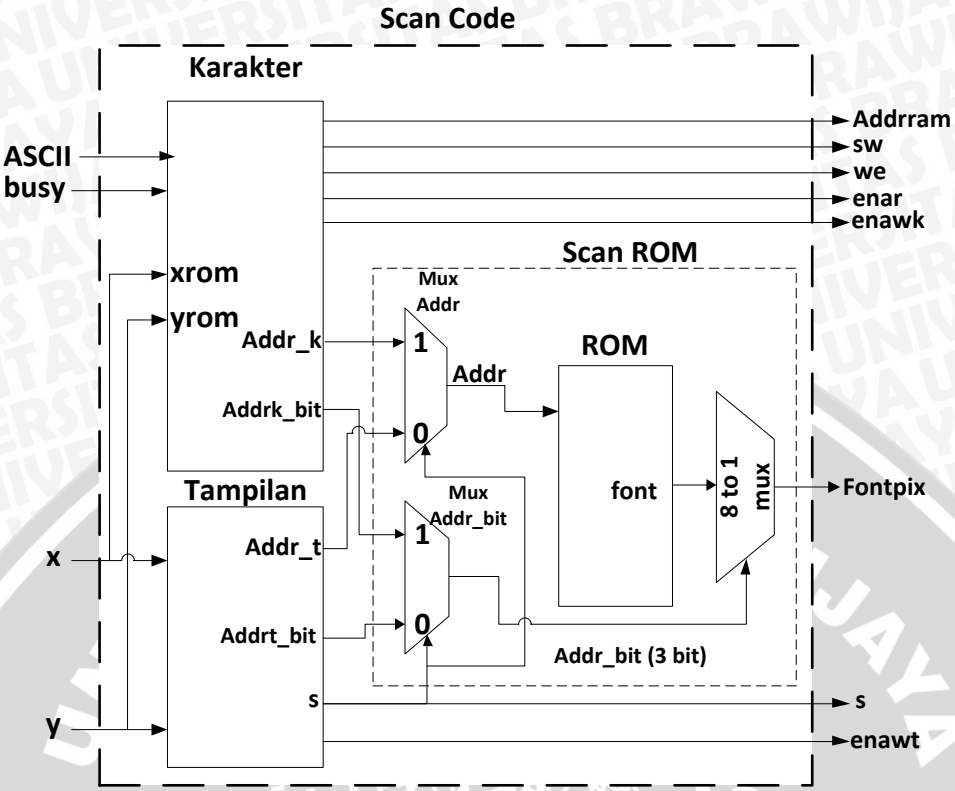
Sistem modul *scan code* ditunjukkan dalam Gambar 4.5 dirancang berdasarkan masukan kode ASCII dan *scan* pixel VGA monitor. Prinsip kerja sistem modul *scan code* adalah membaca data karakter ROM yang direpresentasikan sebagai data pixel. Masukan *pin* modul *scan code* adalah clk, rst, ASCII, *busy*, x, y dan keluaran modul *scan code* yaitu fontpix, addrram, sw, we, enar, enawk, s, enawt. Sistem modul *scan code* terdiri dari blok karakter, tampilan, dan *scan* ROM seperti ditunjukkan dalam Gambar 4.6.



Gambar 4.5 Modul *scan code*.

Keterangan:

- clk = masukan sumber sinyal *clock*
- rst = masukan sinyal reset
- *busy* = masukan sinyal proses pembacaan *Keyboard*
- ASCII = masukan sinyal kode ASCII 8 *bit*
- x = masukan sinyal batas alamat *scan* horizontal VGA
- y = masukan sinyal batas alamat *scan* vertikal VGA
- fontpix = keluaran sinyal data pixel *font* VRAM
- addrram = keluaran sinyal alamat untuk RAM
- sw = keluaran sinyal kontrol untuk *enable* RAM
- we = keluaran sinyal *write enable*
- enar = keluaran sinyal *enable read* untuk RAM
- enawk = keluaran sinyal *enable write* blok karakter untuk RAM
- s = keluaran sinyal kontrol *enable write* untuk RAM
- enawt = keluaran sinyal *enable write* blok tampilan untuk RAM



Gambar 4.6 Diagram blok dari *scan code*.

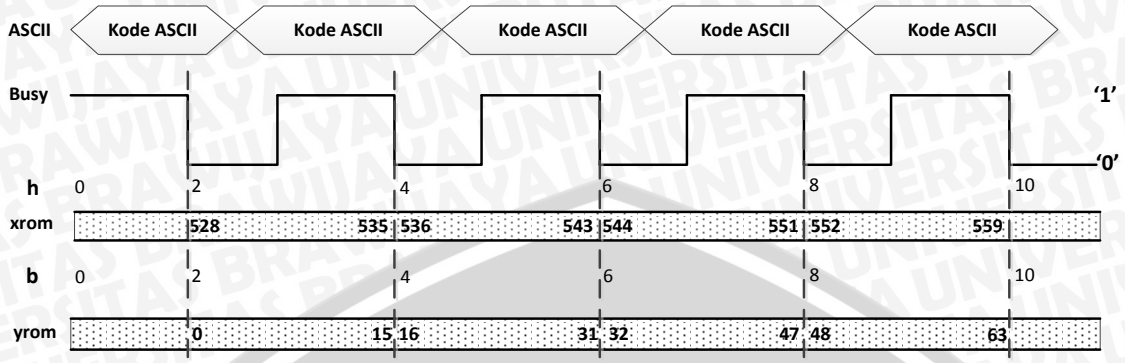
4.2.3.1 Blok Karakter

Kode ASCII diubah sebagai alamat addr ROM pada blok karakter untuk pembacaan karakter ROM. Masukan *pin* blok karakter adalah ASCII, xrom, yrom, *busy* dan keluaran blok karakter adalah addr_k, addrk_*bit*, addrram, sw, we, enar, enawk seperti dalam Gambar 4.6. Dalam blok karakter digunakan proses sekuensial dengan model Moore.

Fungsi masukan kode ASCII dan yrom digunakan sebagai alamat addr_k. Masukan xrom dan yrom juga digunakan sebagai batas pembacaan ROM. Masukan *busy* digunakan sebagai perpindahan batas pembacaan karakter kode ASCII. Dan Nilai addrk_*bit* merupakan nilai 3 *bit* terakhir dari xrom. Fungsi dari masukan dan keluaran masing-masing blok karakter digunakan sebagai proses operasi dan perancangan blok karakter.

Proses pembacaan ROM untuk blok karakter disesuaikan dengan kode ASCII ditunjukkan seperti dalam Gambar 4.7. Jika kode ASCII bernilai “x18” maka nilai b dikurang satu sedangkan jika kode ASCII bernilai “x19” maka nilai b ditambah satu. Jika nilai *busy* berubah dari ‘1’ ke ‘0’ maka nilai h ditambah satu. Nilai b atau baris

digunakan untuk perpindahan baris pada tampilan VGA dan nilai h atau kolom digunakan untuk perpindahan karakter pada tampilan ditunjukkan dalam Gambar 4.8.



Gambar 4.7 *Timing diagram* perancangan blok karakter.

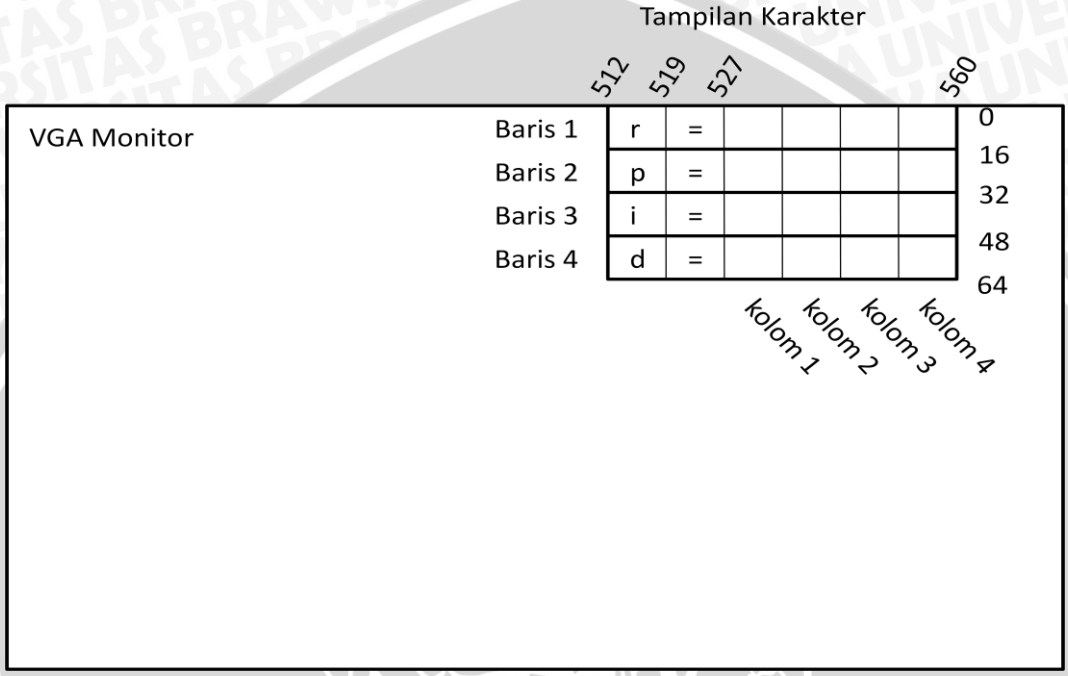
Sinyal addrram dibentuk oleh kombinasi yrom 7 *bit* dan xrom 10 *bit*. Nilai enawk = '1' ketika terjadi proses pembacaan karakter pada ROM sedangkan nilai enawk = '0' ketika tidak terjadi proses pembacaan karakter pada ROM. Nilai sw dan we selalu berubah dari '1' ke '0' maupun '0' ke '1' ketika nilai xrom = 799 dan y = 524. Sinyal enar bernilai '1' ketika nilai we = '0' dan sebaliknya enar bernilai '0' ketika nilai we = '1'.

4.2.3.2 Blok Tampilan

Nilai x dan y diubah sebagai batas alamat pada blok tampilan untuk pembacaan karakter tampilan pada ROM. Masukan pin blok tampilan adalah x, y dan keluaran blok tampilan yaitu addr_t, addrt_*bit*, enawt dan s seperti dalam Gambar 4.6. Nilai x dan y digunakan sebagai batas alamat addr_t. Nilai addrk_*bit* merupakan nilai 3 *bit* terakhir dari x. Nilai s digunakan sebagai pemilih saluran addr dan addr_*bit* dan sebagai keluaran nilai s.

Proses pembacaan untuk blok tampilan ini dilakukan mulai dari batas x = 512 sampai 527 dan batas y = 0 sampai y = 63. Jika x bernilai 512 sampai 519 seperti dalam Gambar 4.8 dan y bernilai 0 sampai 15 ditunjukkan dalam Gambar 4.8 maka pembacaan karakter "r" pada ROM sedangkan x bernilai 520 sampai 527 seperti dalam Gambar 4.8 maka pembacaan karakter "=" pada ROM. Jika x bernilai 512 sampai 519 ditunjukkan dalam Gambar 4.8 dan y bernilai 16 sampai 31 seperti dalam Gambar 4.8 maka pembacaan karakter "p" pada ROM sedangkan x bernilai 520 sampai 527 seperti dalam Gambar 4.8 maka pembacaan karakter "=" pada ROM. Jika x bernilai 512 sampai 519 seperti dalam Gambar 4.8 dan y bernilai 32 sampai 47 ditunjukkan dalam Gambar 4.8 maka pembacaan karakter "i" pada ROM sedangkan x bernilai 520 sampai 527

ditunjukkan dalam Gambar 4.8 maka pembacaan karakter "=" pada ROM. Jika x bernilai 512 sampai 519 ditunjukkan dalam Gambar 4.8 dan y bernilai 48 sampai 63 ditunjukkan dalam Gambar 4.8 maka pembacaan karakter "d" pada ROM sedangkan x bernilai 520 sampai 527 ditunjukkan dalam Gambar 4.8 maka pembacaan karakter "=" pada ROM. Dan jika nilai x bernilai dari 512 sampai 527 dan nilai y ≤ 63 maka nilai s = 0 sedangkan jika tidak maka nilai s = 1.



Gambar 4.8 Rancangan tampilan karakter di VGA monitor.

Sinyal enawt bernilai '1' ketika terjadi pembacaan karakter tampilan pada ROM sedangkan sinyal enawt bernilai '0' ketika tidak terjadi pembacaan karakter tampilan pada ROM. Sinyal s bernilai 0 ketika batas nilai x = 512 sampai x = 527 dan y ≤ 63. Sinyal s bernilai 1 ketika batas nilai x = 528 sampai x = 559 dan y ≤ 63. Sinyal s bernilai 2 ketika batas nilai x ≤ 479 dan y ≤ 63. Sinyal s bernilai 3 ketika batas nilai x ≤ 479 dan y = 64 sampai y = 127. Sinyal s bernilai 4 ketika batas nilai x dan y diluar batas-batas tampilan.

4.2.3.3 Blok *Scan* ROM

Data 8 *bit* ROM dibaca sesuai dengan alamat addr pada blok *scan* ROM. Masukan *pin* blok *scan* ROM adalah addrk_*bit*, addr_k, addrt_*bit*, addr_t, s dan keluaran blok *scan* ROM yaitu fontpix. Blok ROM seperti dalam Gambar 4.6 berisi dengan karakter " ", "r", "p", "i", "d", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "=". Ukuran ROM yang diperlukan dalam blok *scan* ROM sebesar 256 Byte.

Pembacaan data ROM dikontrol oleh nilai s. Jika nilai s = 1 maka nilai addr_*bit* = addrk_*bit* dan addr = addr_k. Dan jika nilai s = 0 maka nilai addr_*bit* = addrt_*bit* dan addr = addr_t. Kemudian data hasil pembacaan ROM 8 *bit* ini direpresentasikan oleh addr_*bit* sebagai fontpix.

4.2.4 Perancangan Sistem Modul *Scan* PWM dan *Rotary*

Sistem modul *scan* PWM dan *Rotary* dirancang berdasarkan besar frekuensi PWM, *Rotary*, dan VGA monitor seperti dalam Gambar 4.9. Frekuensi PWM dan *Rotary* dicacah berdasarkan frekuensi *scan* pixel VGA monitor. Hasil cacahan tersebut digunakan untuk penyesuaian frekuensi tampilan sinyal PWM dan *Rotary* pada VGA monitor. Masukan *pin* modul *scan* PWM dan *Rotary* adalah clk, rst, sinyal_pwm, sinyal_*Rotary*, dan *port* keluarannya adalah puls_pwm dan puls_*Rotary*.

Fungsi masukan sinyal PWM digunakan sebagai bentuk masukan sinyal PWM dengan nilai *duty cycle* tertentu. Fungsi masukan sinyal *Rotary* digunakan sebagai bentuk masukan sinyal *Rotary* dengan nilai frekuensi tertentu. Keluaran Puls_PWM dan Puls_*Rotary* digunakan sebagai bentuk keluaran sinyal PWM dan *Rotary* dengan nilai frekuensi yang telah disesuaikan terlebih dahulu dan tidak lepas dari bentuk sinyal aslinya. Fungsi dari masukan dan keluaran masing-masing modul *scan* PWM dan *Rotary* digunakan sebagai proses operasi dan perancangan modul *scan* PWM dan *Rotary*.

Dalam perancangan sistem modul *scan* PWM dan *Rotary* sangat penting untuk menentukan nilai frekuensi tampilan pada VGA monitor terlebih dahulu. Berdasarkan spesifikasi rancangan, besar nilai frekuensi VGA monitor adalah 25 MHz dan batas tampilan sinyal PWM dan *Rotary* adalah 64x480 pixel. Untuk menentukan batas frekuensi tampilan sinyal dapat dihitung dengan Persamaan (4-1).

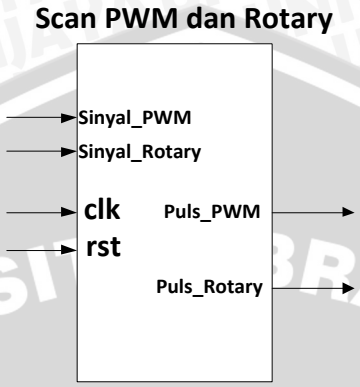
$$\text{frekuensi tampilan} = \frac{\text{frekuensi VGA}}{\text{batas tampilan horizontal sinyal}} \quad (4\text{-}1)$$

$$\text{frekuensi tampilan} = \frac{25\ \text{MHz}}{480}$$

$$\text{frekuensi tampilan} = 52.0833\ \text{KHz}$$

Berdasarkan besar nilai frekuensi tampilan maka besar frekuensi sinyal yang akan ditampilkan pada VGA monitor setidaknya lebih besar atau sama dengan 2 kali besar nilai frekuensi tampilan. Jika besar frekuensi sinyal PWM maupun *Rotary* lebih kecil atau sama dengan frekuensi tampilan maka sinyal PWM dan *Rotary* tidak jelas terlihat pada tampilan VGA monitor. Dan jika besar frekuensi sinyal PWM maupun *Rotary* lebih besar dari frekuensi tampilan maka sinyal PWM dan *Rotary* lebih jelas terlihat

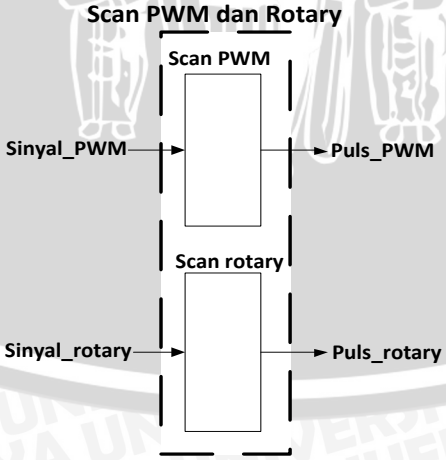
pada tampilan VGA monitor. Dengan syarat tersebut maka dalam perancangan modul *scan* PWM dan *Rotary* ditentukan batas frekuensi sinyal yang akan ditampilkan pada VGA monitor dengan batas minimum 100 KHz dan batas maksimum 12.5 MHz ini dikarenakan besar frekuensi VGA yang digunakan sebesar 25 MHz. Detail perancangan sistem modul *scan* PWM dan *Rotary* ditunjukkan dalam Gambar 4.10.



Gambar 4.9 Modul *scan* PWM dan *Rotary*.

Keterangan:

- clk = masukan sumber sinyal *clock*
- rst = masukan sinyal reset
- Sinyal_PWM = masukan sinyal PWM
- Sinyal_*Rotary* = masukan sinyal *Rotary*
- Puls_PWM = keluaran *scan* sinyal PWM
- Puls_*Rotary* = keluaran *scan* sinyal *Rotary*



Gambar 4.10 Diagram blok dari *scan* PWM dan *Rotary*.

4.2.4.1 Blok *Scan* PWM

Blok *scan* PWM digunakan untuk mencacah sinyal PWM. Besar nilai frekuensi PWM adalah 1 KHz. Besar nilai frekuensi cacah yang digunakan sebesar 25 MHz yang

disesuaikan dengan nilai frekuensi VGA monitor. Dalam perancangan blok *scan* PWM ditentukan sinyal frekuensi keluaran sebesar 1 MHz sebagai frekuensi tampilan pada VGA monitor karena dengan sinyal frekuensi sebesar 1 MHz dapat menampilkan 19 periode pada tampilan sesuai dengan acuan nilai frekuensi tampilan.

Besar frekuensi sinyal PWM 1 KHz dicacah dengan frekuensi 25 MHz. Cacahan dilakukan dengan mencacah frekuensi sinyal PWM 1 KHz. Besar cacahan untuk frekuensi 1 KHz (N_{cacah}) dapat diperhitungkan dalam Persamaan (4-2).

$$N_{cacah} = \frac{\text{Frekuensi Cacah}}{\text{Frekuensi PWM}} \quad (4\text{-}2)$$

$$N_{cacah} = \frac{25\ \text{MHz}}{1\ \text{KHz}}$$

$$N_{cacah} = 25000$$

Dari hasil N_{cacah} tersebut untuk menghasilkan sinyal frekuensi 1 MHz sebagai frekuensi tampilan sinyal PWM maka dapat diperhitungkan dalam Persamaan (4-3).

$$M_{cacah} = \frac{N_{cacah}}{1000} \quad (4\text{-}3)$$

$$M_{cacah} = \frac{25000}{1000}$$

$$M_{cacah} = 25$$

Dari hasil M_{cacah} tersebut menunjukkan bahwa setiap 25 dari 25000 akan menghasilkan frekuensi sinyal 1 MHz dengan frekuensi cacah 25 MHz.

4.2.4.2 Blok *Scan Rotary*

Blok *scan Rotary* digunakan untuk mencacah sinyal *Rotary*. Besar nilai frekuensi cacah yang digunakan sebesar 25 MHz yang disesuaikan dengan nilai frekuensi VGA monitor. Dalam perancangan blok *scan Rotary* ditentukan sinyal frekuensi keluaran sebesar 100 KHz sampai 12.5 MHz sesuai dengan batas frekuensi sinyal yang akan ditampilkan pada VGA monitor.

Dalam perancangan blok *scan Rotary*, nilai cacah dari sinyal *Rotary* ditentukan dengan mencacah nilai frekuensi sinyal *Rotary*. Dengan besar frekuensi sinyal keluaran 100 KHz sampai 12.5 MHz dan frekuensi cacah 25 MHz dapat ditentukan besar batas frekuensi sinyal *Rotary*. Besar cacahan untuk frekuensi sinyal keluaran 100 KHz (Ncacah_*max*) sampai 12.5 MHz (Ncacah_*min*) dapat diperhitungkan dalam Persamaan (4-4) dan Persamaan (4-5).

$$N_{cacah\_max} = \frac{\text{Frekuensi Cacah}}{\text{Frekuensi keluaran } minimum} \quad (4\text{-}4)$$

$$N_{cacah\_max} = \frac{25\ \text{MHz}}{100\ \text{KHz}}$$

$$N_{cacah\_max} = 250$$

$$N_{cacah\ \_min} = \frac{\text{Frekuensi Cacah}}{\text{Frekuensi keluaran } maximum} \quad (4\text{-}5)$$

$$N_{cacah\ \_min} = \frac{25\ \text{MHz}}{12.5\ \text{MHz}}$$

$$N_{cacah\ \_min} = 2$$

Dari hasil N_{cacah_max} dan N_{cacah_min} tersebut untuk menentukan batas frekuensi sinyal *Rotary* sebagai frekuensi tampilan maka dapat diperhitungkan dalam Persamaan (4-6) dan Persamaan (4-7).

$$M_{cacah\ \_max} = N_{cacah\ \_max} \times 1000 \quad (4\text{-}6)$$

$$M_{cacah\ \_max} = 250 \times 1000$$

$$M_{cacah\ \_max} = 250000$$

$$M_{cacah\ \_min} = N_{cacah\ \_min} \times 1000 \quad (4\text{-}7)$$

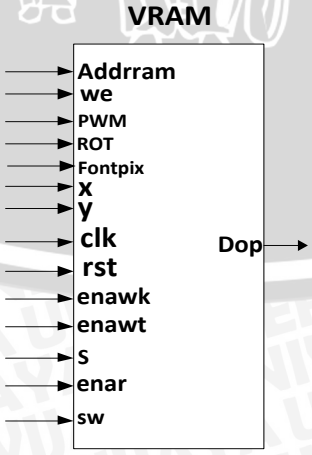
$$M_{cacah\ \_min} = 2 \times 1000$$

$$M_{cacah\ \_min} = 2000$$

Dari hasil M_{cacah_min} dan M_{cacah_max} tersebut menunjukan bahwa dengan nilai 2000 sampai 250000 dan frekuensi cacah 25 MHz maka batas frekuensi sinyal *Rotary* yang bisa dioperasikan sebesar 100 Hz sampai 12.5 KHz.

4.2.5 Perancangan Sistem Modul VRAM

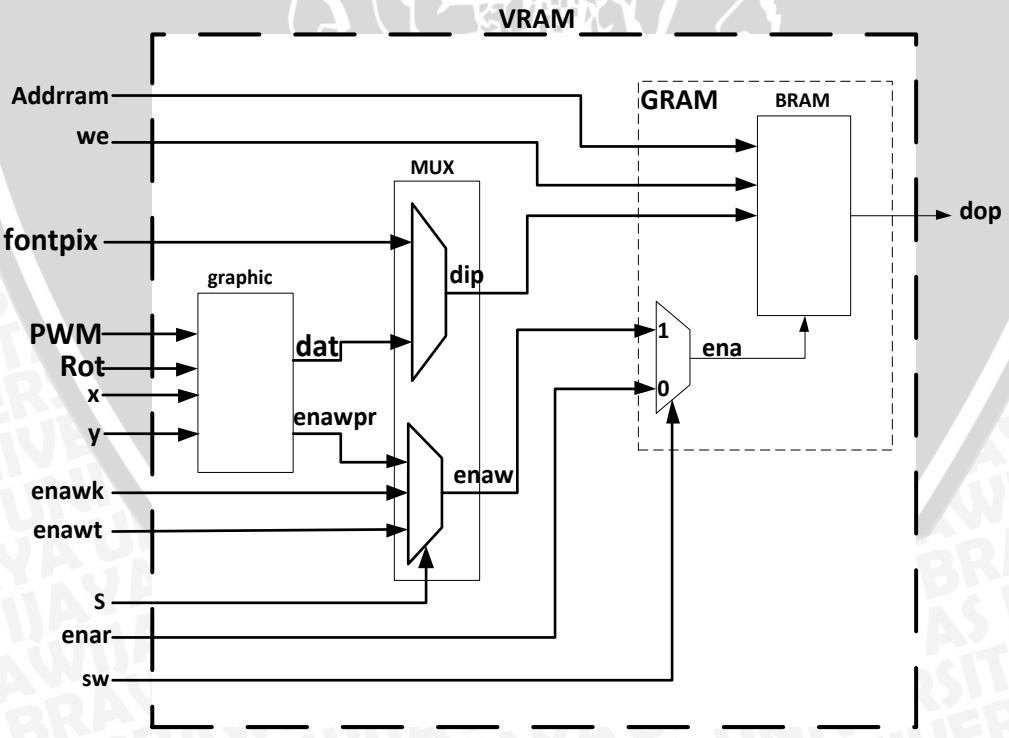
Sistem modul VRAM seperti dalam Gambar 4.11 dirancang berdasarkan masukan *scan* pixel VGA monitor. Prinsip kerja modul VRAM adalah membaca dan menulis data pada memori RAM untuk tampilan VGA monitor. Masukan *pin* modul VRAM adalah clk, rst, pwm, rot, fontpix, x, y, addrram, we, sw, enar, enawk, enawt, s dan sinyal keluaran modul VRAM adalah dop. Sistem modul VRAM terdiri dari blok *Graphic*, MUX, dan GRAM ditunjukkan dalam Gambar 4.12.



Gambar 4.11 Modul VRAM.

Keterangan:

- clk = masukan sumber sinyal *clock*
- rst = masukan sinyal reset
- PWM = masukan sinyal PWM
- ROT = masukan sinyal *Rotary*
- fontpix = masukan sinyal data pixel *font*
- x = masukan sinyal *feedback scan* horizontal VGA
- y = masukan sinyal *feedback scan* vertikal VGA
- we = masukan sinyal *write enable* RAM
- addrram = masukan sinyal alamat RAM
- sw = masukan sinyal kontrol *enable* RAM
- we = masukan sinyal *write enable*
- enar = masukan sinyal *enable read* untuk RAM
- enawk = masukan sinyal *enable write* blok karakter
- s = masukan sinyal kontrol *enable write* RAM
- enawt = masukan sinyal *enable write* blok tampilan
- dop = keluaran sinyal data VRAM

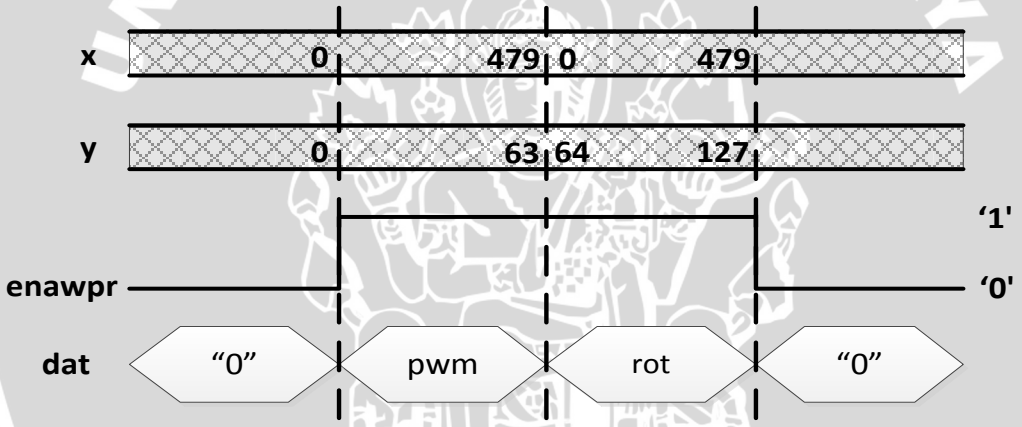


Gambar 4.12 Diagram blok dari VRAM.

4.2.5.1 Blok *Graphic*

Blok *Graphic* digunakan untuk mengolah data masukan yang akan disimpan pada BRAM dengan alamat tertentu. Masukan *pin Graphic* adalah pwm, rot, x, y dan keluaran blok *Graphic* yaitu dat dan enawpr. Masukan PWM digunakan sebagai representasi sinyal PWM. *Pin* ROT digunakan sebagai representasi sinyal *Rotary*. *Pin* x dan y digunakan sebagai batas alamat penulisan data PWM dan *Rotary*. Keluaran sinyal dat digunakan sebagai representasi data PWM dan *Rotary*. Keluaran enawpr digunakan sebagai sinyal penulisan data PWM dan *Rotary* .

Blok *Graphic* dirancang dengan proses sekuensial model Moore dan sesuai dengan *timing diagram* seperti ditunjukkan dalam Gambar 4.13. Nilia sinyal enawpr akan bernilai '1' ketika nilai x ≤ 479 dan y ≤ 127 pada blok *Graphic*. Jika nilai enawp = '1' maka keluaran sinyal dat merupakan data pwm ataupun data *Rotary*. Jika nilai enawpr = '0' maka keluaran sinyal dat bernilai 0.



Gambar 4.13 *Timing diagram* perancangan blok *Graphic*.

4.2.5.2 Blok MUX

Blok MUX digunakan untuk memilih saluran data dan enaw sesuai kondisi sinyal s. Masukan *pin* blok MUX adalah dat, fontpix, enawpr, enawt, enawk dan keluaran blok MUX yaitu dip dan enaw seperti dalam Gambar 4.12. Proses pemilihan saluran masukan blok MUX tersebut dilakukan oleh sinyal kontrol s. Jika nilai s = 0 maka keluaran dip adalah masukan fontpix dan keluaran enaw adalah masukan enawt. Jika nilai s = 1 maka keluaran dip adalah masukan fontpix dan keluaran enaw adalah masukan enawk. Jika nilai s = 2 dan s = 3 maka keluaran dip adalah masukan dat dan keluaran enaw adalah masukan enawpr.

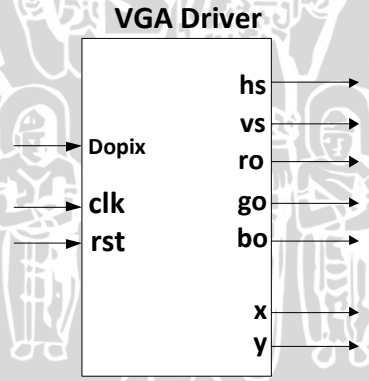
4.2.5.3 Blok GRAM

Blok GRAM digunakan untuk proses pembacaan dan penulisan data pada alamat tertentu. Blok BRAM dirancang dengan menggunakan *Memory Generator Block ipcore logic*. Blok BRAM yang dirancang adalah *single port* RAM seperti dalam Gambar 2.13. Lebar addr pada BRAM yaitu 17 *bit* dan lebar data 1 *bit*. Masukan *pin* blok GRAM adalah addrram, sw, we, dip, enaw, enar dan keluaran blok GRAM yaitu dop seperti dalam Gambar 4.12.

Pada proses blok GRAM, jika sw = '1' maka ena = enaw sedangkan jika sw = '0' maka ena = enar. Ketika we = '1' maka dilakukan proses penulisan sinyal dip pada BRAM sesuai dengan alamat addrram. Dan ketika we = '0' maka dilakukan proses pembacaan sinyal dop pada BRAM sesuai dengan alamat addrram.

4.2.6 Perancangan Sistem Modul VGA *Driver*

Sistem modul VGA *Driver* seperti dalam Gambar 4.14 dirancang dengan berdasarkan masukan data pixel dan *timing diagram scan* pixel ditunjukkan dalam Gambar 2.9 dan Gambar 2.10. Modul VGA *driver* digunakan untuk mengoperasikan data masukan menjadi data pixel VGA monitor. Masukan *pin* modul VGA *driver* adalah clk, rst, dopix dan keluaran modul VRAM adalah hs, vs , ro, go, bo, x, y.



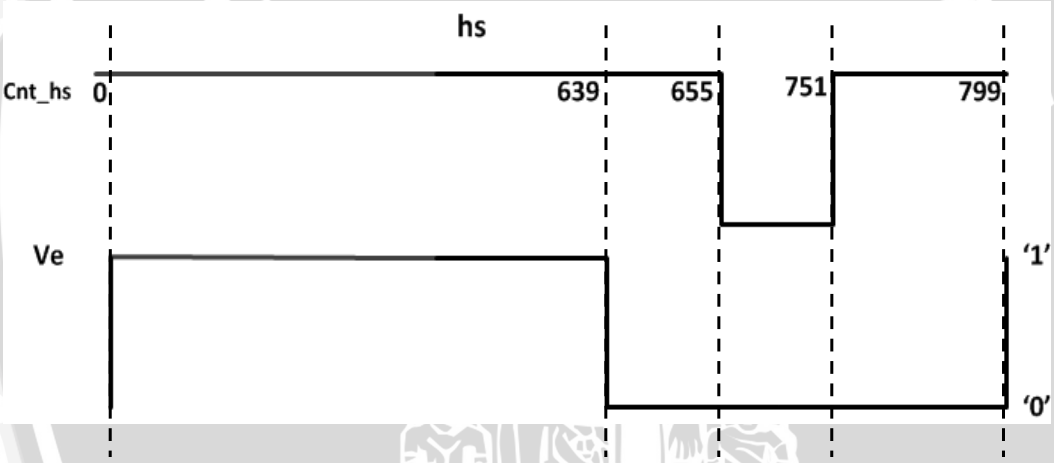
Gambar 4.14 Modul VGA *driver*.

Keterangan:

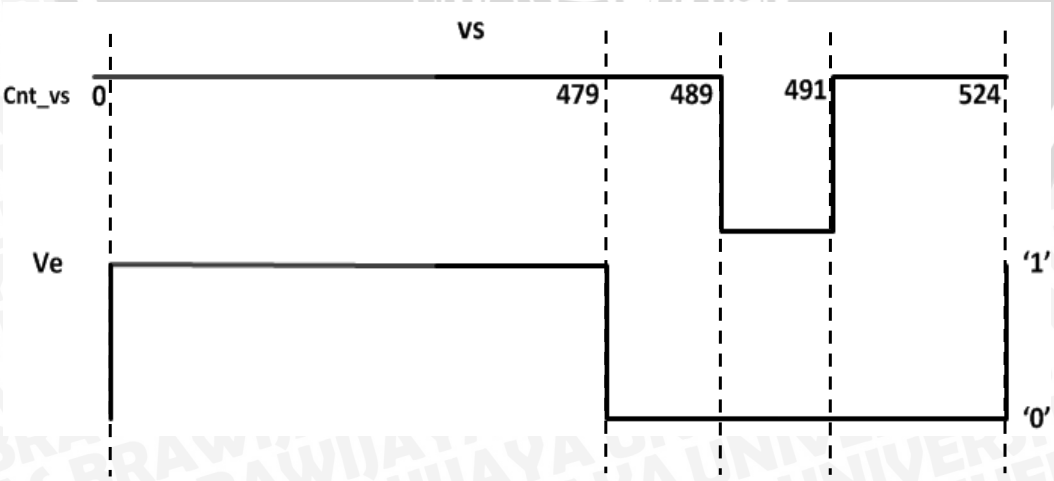
- clk = masukan sumber sinyal *clock*
- rst = masukan sinyal reset
- dopix = masukan sinyal data RAM
- bo = keluaran sinyal *blue color* VGA 2 *bit*
- go = keluaran sinyal *green color* VGA 3 *bit*
- ro = keluaran sinyal *red color* VGA 3 *bit*
- x = keluaran sinyal *feedback scan* horizontal VGA

- y = keluaran sinyal *feedback scan* vertikal VGA
- hs = keluaran sinyal *scan* horizontal VGA
- vs = keluaran sinyal *scan* vertikal VGA

Blok VGA *driver* menggunakan proses sekuensial dengan model Moore. Masukan sinyal dopix direpresentasikan sebagai nilai ro, go, dan bo pada saat nilai ve atau *video enable* bernilai '1' dan ro, go, bo bernilai '0' ketika nilai ve = '0'. Jika nilai cnt_hs atau *scan* horizontal ≤ 639 dan cnt_vs atau *scan* vertikal ≤ 479 maka ve = '1' sedangkan jika cnt_hs ≥ 639 atau cnt_vs ≥ 479 maka ve = '0'. Jika nilai cnt_hs ≥ 656 dan cnt_hs ≤ 751 ditunjukkan dalam Gambar 4.15 maka nilai hs = '0' sedangkan jika tidak maka nilai hs = '1'. Jika cnt_vs ≥ 490 dan cnt_vs ≤ 491 ditunjukkan dalam Gambar 4.16 maka nilai vs = '0' sedangkan jika tidak maka nilai vs = '1'. Keluaran nilai x dan y merupakan nilai cnt_hs dan cnt_vs secara singkron.



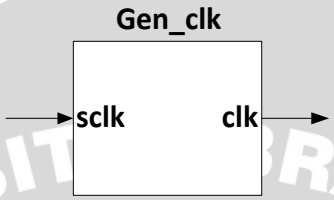
Gambar 4.15 *Timing diagram scan* horizontal perancangan blok VGA.



Gambar 4.16 *Timing diagram scan* vertikal perancangan blok VGA.

4.2.7 Perancangan Sistem Modul Gen_clk

Sistem modul Gen_clk seperti dalam Gambar 4.17 dirancang untuk menghasilkan keluaran *clock* 25 MHz. Sistem modul Gen_clk dirancang oleh *Digital Clock Manager* (DCM) sebagai *clock devider* dengan masukan *clock* eksternal 50 MHz. Nilai sinyal *clock* 25 MHz sangat dibutuhkan pada VGA *driver* karena frekuensi VGA monitor yang digunakan pada sistem ini adalah 25 MHz sesuai dengan Persamaan (2-4). Sinyal *clock* 25 MHz juga diberikan pada sistem lainnya karena untuk singkronisasi antar sistem.



Gambar 4.17 Modul Gen_clk.

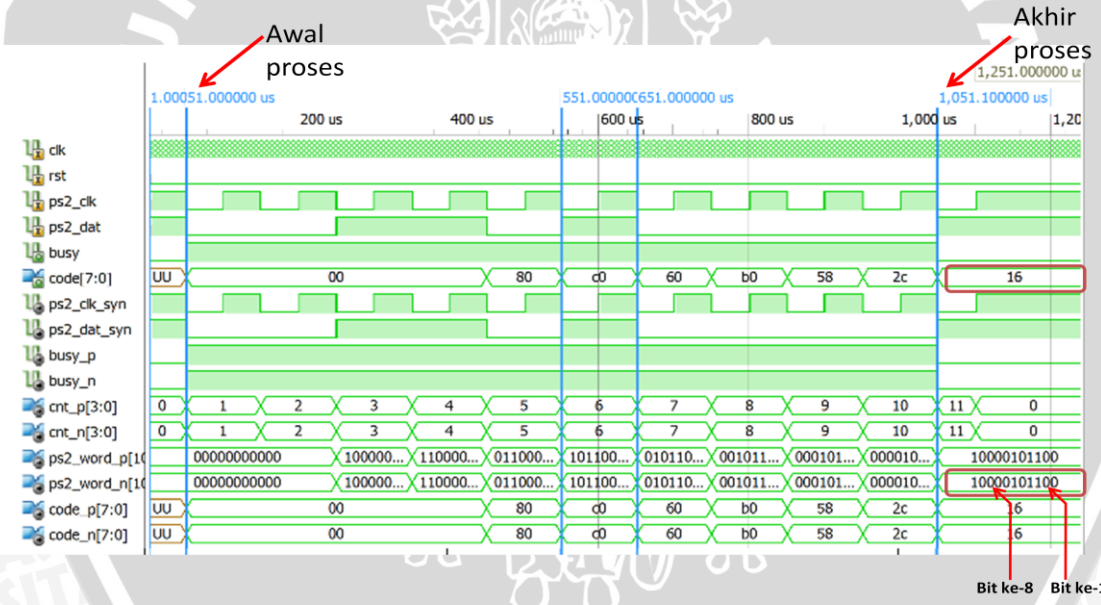
BAB V
HASIL DAN PEMBAHASAN

5.1 Simulasi dan Pengujian Sistem Pemonitoran

Pada simulasi diberikan nilai clk dengan periode 40 ns (frekuensi 25 MHz) dan *duty cycle* sebesar 50 %. Nilai rst selalu diberikan dengan nilai '0' dalam simulasi. Simulasi dan pengujian ini dilakukan untuk mengetahui kecepatan proses dan kinerja modul dalam mengolah masukan menjadi keluaran.

5.1.1 Modul *Keyboard Driver*

Pada modul *Keyboard driver*, tombol yang disimulasikan adalah tombol '1' dengan kode "00110100011". Pada hasil simulasi ditunjukkan dalam Gambar 5.1 menunjukkan bahwa nilai keluaran *code* "x16" sesuai dengan 8 *bit* (*bit* ke-1 sampai ke-8) nilai masukan ps2_dat. Dan waktu yang dibutuhkan untuk memproses masukan *Keyboard* menjadi *make code* tanpa adanya *break code* atau *command code* adalah 1000,1 µs.



Gambar 5.1 Hasil simulasi *waveform* modul *Keyboard driver*.

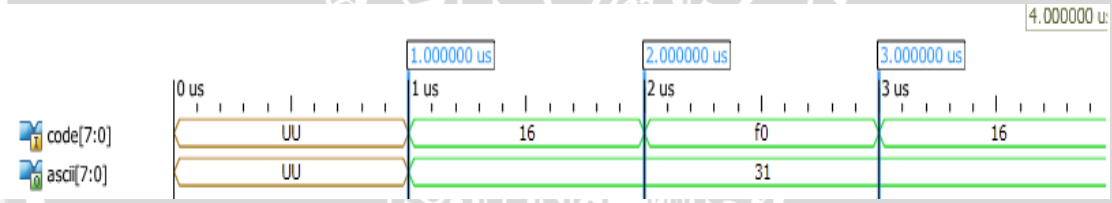
Pada pengujian *Keyboard driver*, tombol *Keyboard* yang diuji dengan karakter 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *backspace*, *enter*, *Up arrow* dan *Down arrow*. Pada pengujian modul *Keyboard driver* diberikan masukan tombol *Keyboard* dengan karakter "1" maka hasil yang dikeluarkan pada modul *Keyboard driver* adalah *make code* nilai "x16". Pada hasil pengujian modul *Keyboard driver* tersebut dapat menunjukkan bahwa data keluaran *make code* sudah sesuai dengan masukan karakter tombol *Keyboard* dan *make code Keyboard* ditunjukkan dalam Gambar 2.7. Hasil pengujian modul *Keyboard driver* dapat ditunjukkan dalam Tabel 5.1.

Tabel 5.1 Hasil pengujian modul *Keyboard driver*

No.	Karakter atau *Command*	*Make Code* (Led(7:0))	Hex *Code*
1	0	01000101	45
2	1	00010110	16
3	2	00011110	1E
4	3	00100110	26
5	4	00100101	25
6	5	00101110	2E
7	6	00110110	36
8	7	00111101	3D
9	8	00111110	3E
10	9	01000110	46
11	*Backspace*	01100110	66
12	*Enter*	01011010	5A
13	*Upper Arrow*	01110101	75
14	*Down Arrow*	01110010	72

5.1.2 Modul *Keyboard Converter*

Pada modul *Keyboard converter*, *make code* yang disimulasikan dengan nilai *code* "x16". Pada hasil simulasi ditunjukkan dalam Gambar 5.2 yang menunjukkan bahwa nilai keluaran ascii "x31" sesuai dengan masukan *code* "x16". Dan waktu yang dibutuhkan sebesar 0 ns untuk memproses masukan *code* menjadi keluaran ascii.



Gambar 5.2 Hasil simulasi *waveform* modul *Keyboard converter*.

Pada modul *Keyboard converter*, *make code* yang diuji adalah "x45", "x16", "x1E", "x26", "x25", "x2E", "x36", "x3D", "x3E", "x46", "x66", "x5A", "x75" dan "x72". Pada masukan *make code* dengan nilai "x16" maka data ascii yang dikeluarkan pada modul *Keyboard converter* dengan nilai "x31". Pada hasil pengujian modul *Keyboard converter* tersebut dapat menunjukkan bahwa data keluaran ascii sudah sesuai dengan masukan *make code* dan tabel kode ASCII yang ditunjukkan dalam Tabel 2.1. Hasil pengujian modul *Keyboard converter* ini ditunjukkan dalam Tabel 5.2.

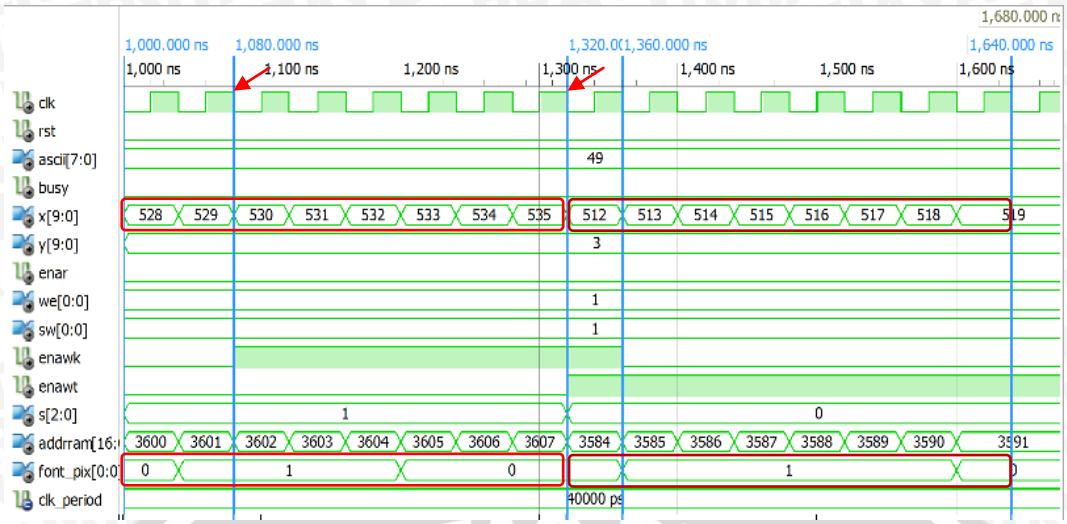
Tabel 5.2 Hasil pengujian modul *Keyboard converter*

No.	Karakter atau *Command*	*Make Code* (*Switch*(7:0))	Hex *Make Code*	ASCII *Code* (Led(7:0))	Hex ASCII *Code*
1	0	01000101	45	00110000	30
2	1	00010110	16	00110001	31
3	2	00011110	1E	00110010	32
4	3	00100110	26	00110011	33
5	4	00100101	25	00110100	34
6	5	00101110	2E	00110101	35
7	6	00110110	36	00110110	36
8	7	00111101	3D	00110111	37
9	8	00111110	3E	00111000	38
10	9	01000110	46	00111001	39
11	*Backspace*	01100110	66	00001000	08
12	*Enter*	01011010	5A	00001101	0D
13	*Upper Arrow*	01110101	75	00011000	18
14	*Down Arrow*	01110010	72	00011001	19

5.1.3 Modul *Scan Code*

Pada modul *scan code*, data ASCII yang disimulasikan adalah “x31”. Pada hasil simulasi ditunjukkan dalam Gambar 5.3 yang menunjukkan bahwa nilai keluaran fontpix sesuai dengan data ROM pada alamat tertentu. Dengan nilai ASCII “x31”, x bernilai 528 sampai 535 dan yrom = 3 maka alamat ROM bernilai 99 dengan keluaran sinyal fontpix berturut-turut adalah “0”, ”1”, ”1”, ”1”, ”1”, ”0”, ”0”, ”0” dan nilai sinyal enawk = ‘1’. Dengan nilai x bernilai 512 sampai 519 dan y = 3 maka alamat ROM bernilai 19 dengan keluaran fontpix berturut-turut adalah “0”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1” ,”0” dan nilai sinyal enawt = ‘1’. Pada hasil simulasi tersebut waktu yang dibutuhkan untuk membaca satu *array* ROM sebesar 320 ns. Dan pada waktu 1080 ns menunjukkan keluaran sinyal enawk terlambat sebesar 80 ns.

Pada modul *scan code*, karakter yang diuji adalah “1” dengan memberikan nilai addr ROM mulai “x60” sampai “x6F”. Pada addr ROM dengan nilai “x63”, data keluaran yang terbaca dengan nilai “01111000”. Pada hasil pengujian modul *scan code* dengan nilai addr ROM tersebut dapat menunjukkan bahwa data keluaran sudah sesuai dengan bentuk karkater yang diuji pada modul *scan code*. Hasil pengujian modul *scan code* dapat ditunjukkan dalam Tabel 5.3.



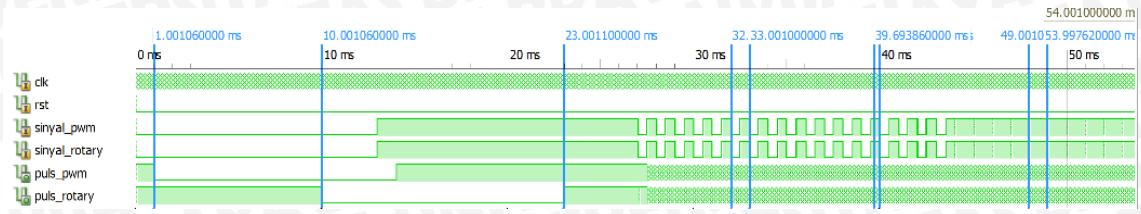
Gambar 5.3 Hasil simulasi *waveform* modul *scan code*.

Tabel 5.3 Hasil pengujian ROM modul *scan code*

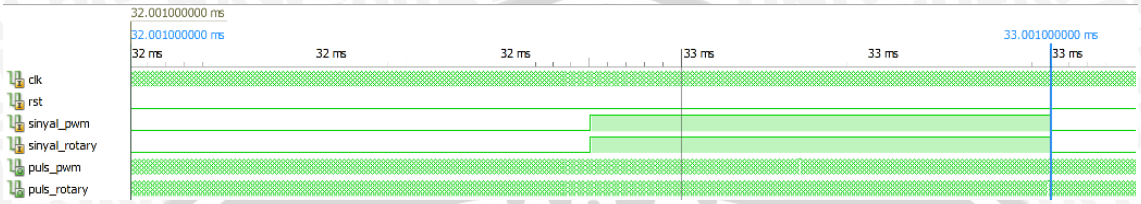
No.	Array ke-	addr_rom (*Switch*(7:0))	addr_rom	word_font (Led(7:0))
1	96	01100000	60	00000000
2	97	01100001	61	00000000
3	98	01100010	62	00111000
4	99	01100011	63	01111000
5	100	01100100	64	11011000
6	101	01100101	65	00011000
7	102	01100110	66	00011000
8	103	01100111	67	00011000
9	104	01101000	68	00011000
10	105	01101001	69	00011000
11	106	01101010	6A	00011000
12	107	01101011	6B	111111111
13	108	01101100	6C	00000000
14	109	01101101	6D	00000000
15	110	01101110	6E	00000000
16	111	01101111	6F	00000000

5.1.4 Modul *Scan* PWM dan *Rotary*

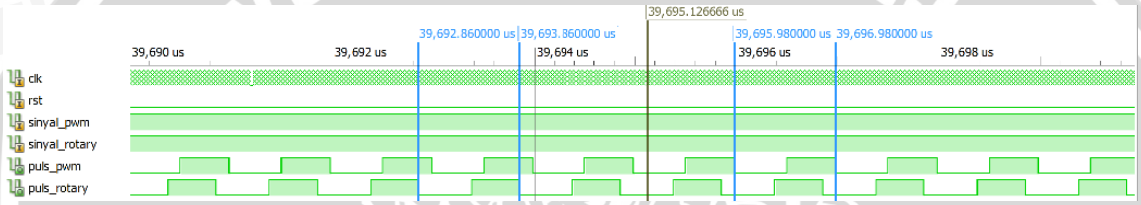
Pada modul *scan* PWM dan *Rotary*, sinyal PWM dan *Rotary* dimulasikan sebagai data masukan "0" dan "1" dengan periode 1 ms. Hasil simulasi sepeti dalam Gambar 5.4 menunjukkan bahwa sinyal PWM dan sinyal *Rotary* dengan kondisi "0" ataupun "1" maka keluaran sinyal juga akan bernilai "0" ataupun "1". Pada hasil simulasi dengan sinyal masukan 1 ms dengan duty cycle 50% ditunjukkan dalam Gambar 5.5 maka hasil keluaran menunjukkan sinyal seperti sinyal aslinya dengan periode 1 µs dengan *duty cycle* 50% yang ditunjukkan dalam Gambar 5.6.



Gambar 5.4 Hasil simulasi *waveform* modul *scan* PWM dan *Rotary*.



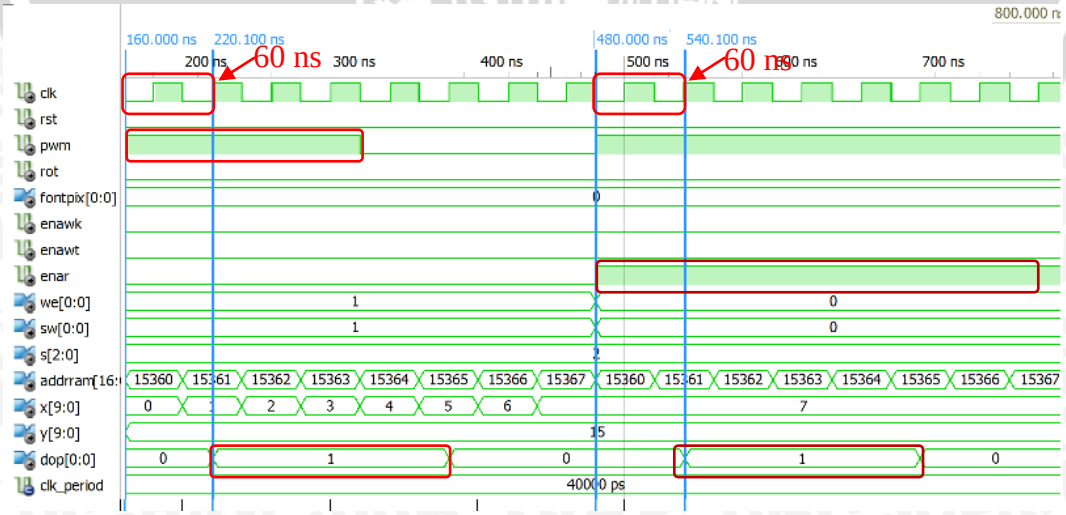
Gambar 5.5 Hasil simulasi *waveform* modul *scan* PWM dan *Rotary zoom input*.



Gambar 5.6 Hasil simulasi *waveform* modul *scan* PWM dan *Rotary zoom output*.

5.1.5 Modul VRAM

Pada modul VRAM, data PWM disimulasikan sebagai data masukan "1" dan "0". Hasil simulasi ditunjukkan dalam Gambar 5.7 yang menunjukkan bahwa data PWM dapat ditulis dan dibaca pada BRAM. Waktu yang dibutuhkan untuk proses tulis dan baca data PWM pada RAM sebesar 60 ns.



Gambar 5.7 Hasil simulasi *waveform* modul VRAM.

Pada modul VRAM, alamat yang diuji adalah alamat dengan ukuran 6 *bit* yang ditentukan secara acak. Masukan alamat diberikan nilai "0000000" dan masukan data diberikan nilai "1". Ketika tombol wr ditekan maka data "1" tertulis pada alamat "0000000" dan ketika tombol wr tidak ditekan maka data yang terbaca pada alamat "0000000" dengan nilai "1". Pada hasil pengujian modul VRAM tersebut dapat menunjukkan bahwa data masukan dapat ditulis dan dibaca pada alamat yang telah ditentukan pada modul VRAM. Hasil pengujian modul VRAM dapat ditunjukkan dalam Tabel 5.4.

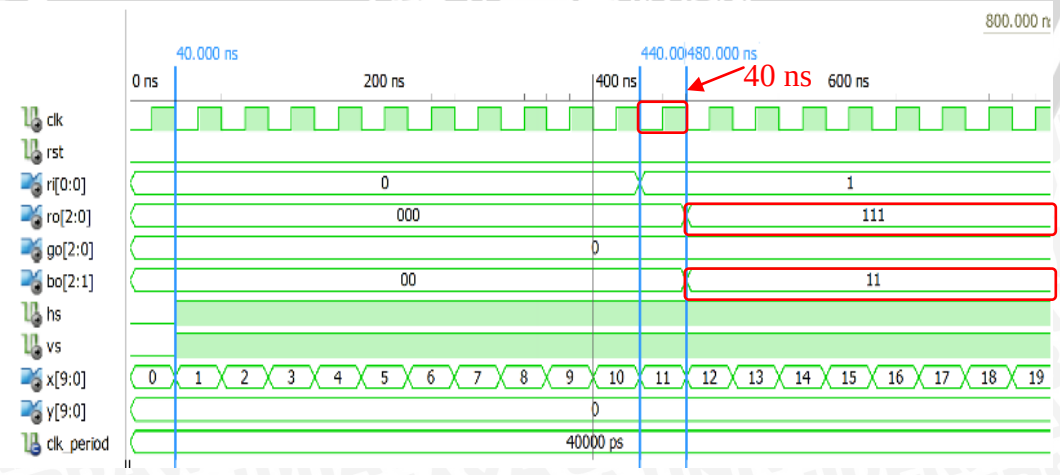
Tabel 5.4 Hasil pengujian modul VRAM

No.	wr (Button(4))	Addr VRAM (*Switch*(6:0)) y(6:6)	Addr VRAM (*Switch*(6:0)) x(5:0)	dip (*Switch*(7:7))	dop (Led(0:0))
1	1	0	000000	1	-
2			000001	1	-
3			000010	0	-
4			000011	0	-
5			000100	0	-
6			001000	1	-
7			010001	0	-
8			100101	1	-
9		1	000000	0	-
10			000001	0	-
11			000010	0	-
12			000011	1	-
13			000100	1	-
14			001000	0	-
15			010001	1	-
16			100101	0	-
17	0	0	000000	-	1
18			000001	-	1
19			000010	-	0
20			000011	-	0
21			000100	-	0
22			001000	-	1
23			010001	-	0
24			100101	-	1
25		1	000000	-	0
26			000001	-	0
27			000010	-	0
28			000011	-	1
29			000100	-	1
30			001000	-	0
31			010001	-	1
32			100101	-	0

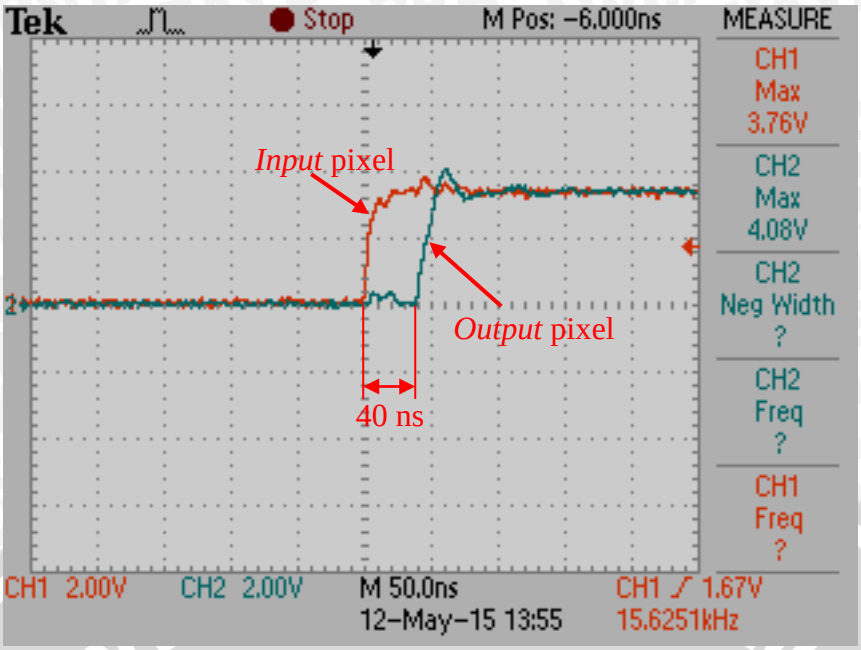
5.1.6 Modul VGA *Driver*

Pada modul VGA *driver*, data r disimulasikan sebagai data masukan VGA *driver*. Pada hasil simulasi ditunjukkan dalam Gambar 5.8 yang menunjukkan bahwa data r dengan nilai "1" dapat dikeluarkan sebagai data ro dengan nilai "111" dan bo dengan nilai "111", sehingga keluaran ro dan bo sudah sesuai dengan masukan r. Waktu yang dibutuhkan oleh VGA *driver* untuk memproses masukan data pixel r menjadi keluaran RGB sebesar 40 ns. Waktu proses 40 ns ini juga dibuktikan dari hasil pengujian modul VGA *driver* yang disesuaikan dengan masukan nilai r pada simulasi. Kecepatan proses sistem diuji pada modul VGA *driver* ini dengan menggunakan Oscilloscope untuk melihat kecepatan proses modul VGA *driver* ditunjukkan dalam Gambar 5.9. *Channel* 2 direpresentasikan sebagai sinyal *output* ro, *Channel* 1 direpresentasikan sebagai sinyal *input* r, dan *time/div* diberikan sebesar 50 ns maka hasil uji modul VGA *driver* menunjukkan kecepatan proses sistem modul VGA *driver* sebesar 40 ns.

Pada pengujian modul VGA *driver*, data yang diuji berupa masukan RGB 8 *bit* sebagai warna tampilan VGA monitor. Masukan 3 *bit* Ri diberikan nilai "111", 3 *bit* Gi diberikan nilai "000" dan 2 *bit* Bi diberikan nilai "00" sehingga pada VGA monitor menunjukkan warna merah. Hasil pengujian modul VGA *driver* tersebut dapat menunjukkan bahwa warna tampilan VGA monitor sesuai dengan masukan 8 *bit* pada modul VGA *driver*. Dan hasil pengujian modul VGA *driver* tersebut sudah sesuai dengan ketentuan kombinasi warna VGA monitor ditunjukkan dalam Gambar 2.8. Hasil pengujian modul VGA *driver* ini dapat ditunjukkan dalam Tabel 5.5.



Gambar 5.8 Hasil simulasi *waveform* modul VGA *driver*.



Gambar 5.9 Hasil pengujian kecepatan proses sistem modul VGA *driver*.

Tabel 5.5 Hasil pengujian modul VGA *driver*

No.	Input Pixel (*Switch*(7:0))			RGB (Led(7:0))	Warna VGA Monitor
	Bi(7:6)	Gi(5:3)	Ri(2:0)		
1	00	000	000	00000000	Hitam
2	00	000	111	00000111	Merah
3	00	111	000	00111000	Hijau
4	00	111	111	00111111	Kuning
5	11	000	000	11000000	Biru
6	11	000	111	11000111	Ungu
7	11	111	000	11111000	Biru Muda
8	11	111	111	11111111	Putih

5.2 Hasil Sintesis Sistem Pemonitoran

Sintesis dilakukan pada setiap modul penyusun sistem pemonitoran untuk mengetahui jumlah penggunaan komponen logika. Hasil sintesis pada modul *Keyboard driver* menunjukkan jumlah komponen yang digunakan adalah 1 *Adder*, 39 *Register*, 2 *Comparator*. Jumlah komponen yang digunakan pada hasil sintesis modul *Keyboard converter* adalah 14 *Comparator*. Jumlah komponen yang digunakan pada hasil sintesis modul *Scan code* adalah 1 256x8-*bit* ROM, 2 *Adder*, 1 *Subtractor*, 25 *Register*, 39 *Comparator*, dan 1 *Multiplexer*. Pada hasil sintesis modul *Scan* PWM dan *Rotary*, jumlah komponen yang digunakan adalah 6 *Adder*, 2 *Counter*, 84 *Register*, dan 8 *Comparator*. Jumlah komponen yang digunakan pada hasil sintesis modul VRAM adalah 2 *Register*, dan 4 *Comparator*. Jumlah komponen yang digunakan pada hasil

sintesis modul VGA adalah 1 *Adder*, 1 *Counter*, 17 *Register*, dan 6 *Comparator*. Dan hasil sintesis total sistem pemonitoran menunjukkan bahwa penggunaan sumber daya tidak melebihi kapasitas yang disediakan oleh Nexys2. Dengan demikian implementasi FPGA sebagai sistem pemonitoran bisa dilakukan pada sistem Xilinx Spartan 3E Nexys2. Hasil sintesis total ini ditunjukkan dalam Tabel 5.6.

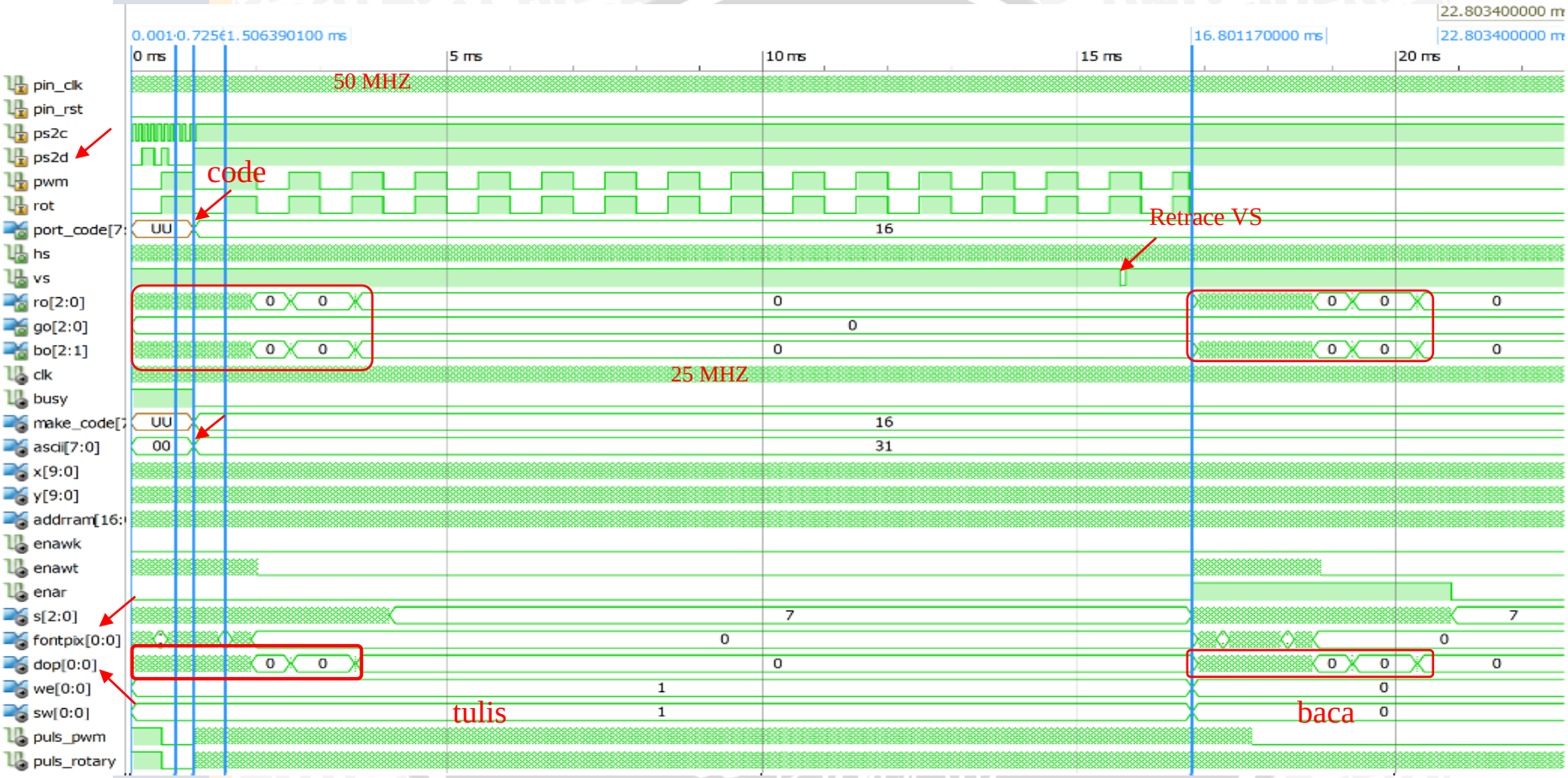
Tabel 5.6 Hasil sintesis sistem pemonitoran

Penggunaan Logika	Jumlah digunakan	Penggunaan
Slice register	97 dari 9312	1%
4 *input* LUTs	395 dari 9312	4%
Occupied Slice	236 dari 4656	5%
Bonded IOBs	24 dari 232	10%
RAMB16s	8 dari 20	40%
BUFGMUXs	2 dari 24	8%
DCMs	1 dari 4	25%

5.3 Simulasi dan Pengujian Keseluruhan Sistem Pemonitoran

Masukan yang disimulasikan pada sistem pemonitoran dengan memberikan data masukan ps2d dengan kode “00110100011”, PWM dan *Rotary* dengan waktu periode 1 ms. Hasil simulasi sistem pemonitoran ditunjukkan dalam Gambar 5.10 yang menunjukkan bahwa masukan data ps2d dapat dioperasikan dengan keluaran “x16” dan data PWM dapat tertulis pada BRAM sehingga keluaran RGB sesuai dengan keluaran data RAM. Dengan demikian hasil simulasi keseluruhan sistem pemonitoran sudah sesuai dengan hasil simulasi tiap-tiap modul sistem pemonitoran.

Pada pengujian modul sistem pemonitoran, data yang diuji berupa tombol *Keyboard* dan *switch* 2 *bit* yang merepresentasikan sinyal PWM dan *Rotary* sebagai tampilan karakter dan sinyal pada VGA monitor. Ketika tombol “0” ditekan pada *Keyboard* maka hasil tampilan VGA monitor menunjukkan karakter “0”. Ketika nilai *switch* sebagai representasi sinyal PWM dan *Rotary* bernilai ‘0’ maka hasil tampilan sinyal PWM dan *Rotary* menunjukkan “*LOW* ” pada VGA monitor sedangkan jika *switch* bernilai ‘1’ maka hasil tampilan menunjukkan “*HIGH*” pada VGA monitor. Hasil pengujian modul sistem pemonitoran tersebut dapat menunjukkan bahwa tampilan karakter dan sinyal serta pada VGA monitor sudah sesuai dengan masukan tombol *Keyboard* dan *switch*. Hasil pengujian modul sistem pemonitoran ini dapat ditunjukkan dalam Tabel 5.7 dan Tabel 5.8.



Gambar 5.10 Hasil simulasi *waveform* sistem pemonitoran.

Tabel 5.7 Hasil pengujian *Keyboard* modul sistem pemonitoran

No.	Tombol *Keyboard*	Tampilan VGA Monitor
1	0	0
2	1	1
3	2	2
4	3	3
5	4	4
6	5	5
7	6	6
8	7	7
9	8	8
10	9	9
11	*Backspace*	Karakter terhapus
12	*Enter*	Karakter terkonfigurasi
13	*Upper Arrow*	Pindah baris atas
14	*Down Arrow*	Pindah baris bawah

Tabel 5.8 Hasil pengujian sinyal PWM dan *Rotary* modul sistem pemonitoran

No.	*Switch* (1:0)	Tampilan VGA Monitor	Sinyal
1	*Switch*(0) = '0'	*LOW*	PWM
	Switch(0) = '1'	*HIGH*	
2	*Switch*(1) = '0'	*LOW*	*Rotary*
	Switch(1) = '1'	*HIGH*	

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan dan pengujian yang dilakukan pada sistem pemonitoran dapat ditarik kesimpulan yang merujuk pada rumusan masalah penelitian ini yaitu:

1. Besar spesifikasi kecepatan VGA monitor yang digunakan untuk sistem pemonitoran adalah 40 ns atau 25 MHz.
2. Sistem pemonitoran bekerja dengan baik menggunakan proses sistem sekuensial dan kombinasional, serta antar modul penyusun sistem terdapat komunikasi data secara paralel maupun serial.
3. Sistem pemonitoran bekerja dengan susunan arsitektur sistem yaitu *Keyboard Driver*, *Keyboard Converter*, *Scan Code*, *scan* PWM dan *Rotary*, VRAM dan VGA *Driver*.
4. Implementasi sistem pemonitoran pada modul FPGA Nexys2 dengan penggunaan sumber daya terbesar pada BRAM sebesar 40% atau sejumlah 8 dari 20.

6.2 Saran

Pada penelitian ini perlu dilakukan penyempurnaan dan pengembangan untuk sistem pemonitoran ini sebagai berikut:

1. Frekuensi *clock* modul penyusun sistem pemonitoran dapat ditingkatkan sebesar 100 MHz kecuali modul VGA *driver*.
2. Proses aritmatika dapat dibuat dengan menggunakan paket DSP atau *Math Function*.
3. Dapat ditambahkan sistem modul untuk menampilkan data nilai *Rotary* dalam satuan rps ataupun rpm pada tampilan monitor.
4. Dapat diimplementasikan ke modul FPGA yang menyediakan sumber daya lebih banyak daripada Nexys2.

DAFTAR PUSTAKA

Chu, P. P. 2008. *FPGA PROTOTYPING BY VHDL EXAMPLES.* New Jersey: John Wiley & Sons.

Digilent. 2011. *Digilent Nexys2 Board Reference Manual.* Washington: Digilent.

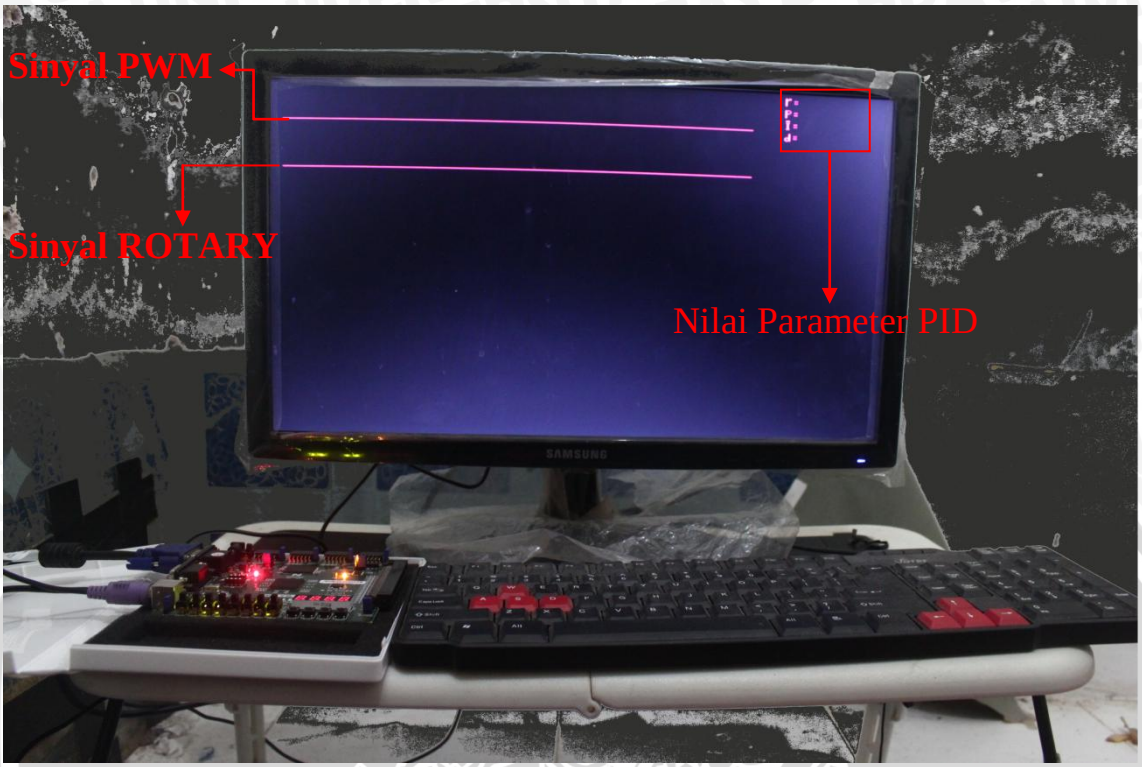
Jogalekar, K., Gunjal, A., Sonawane, S., & Sonawane, D. N. 2013. Implementation of PID Architecture In FPGA for DC Motor Speed Control. *Circuits, Controls and Communications (CCUBE), 2013 International conference on*, 1 - 5.

Xilinx. 2011. *LogicCORE IP BLOCK MEMORY GENERATOR v6.1.* USA: Xilinx.

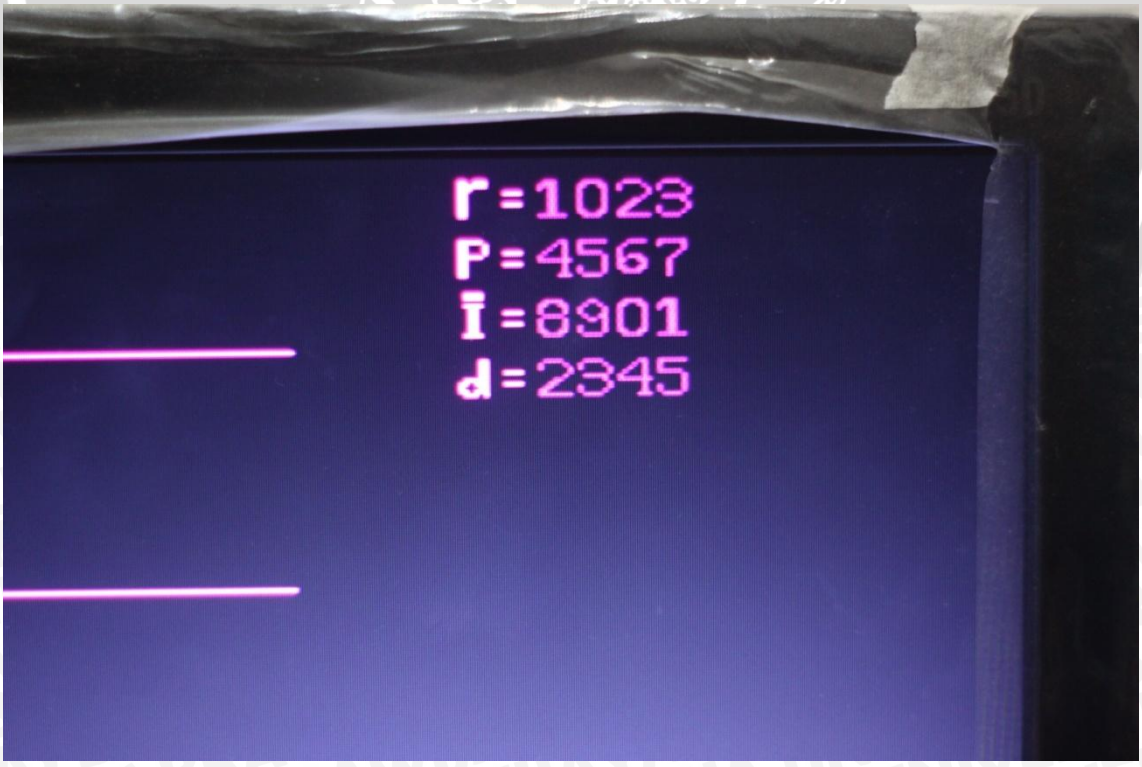
Xilinx. 2013. *Introduction to FPGA Design with Vivado High-Level Synthesis.* USA: Xilinx.

LAMPIRAN

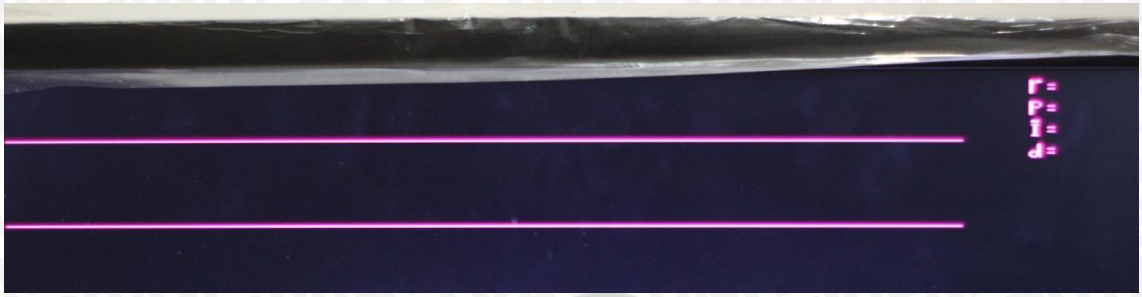
Lampiran 1 Foto Pengujian dan Alat



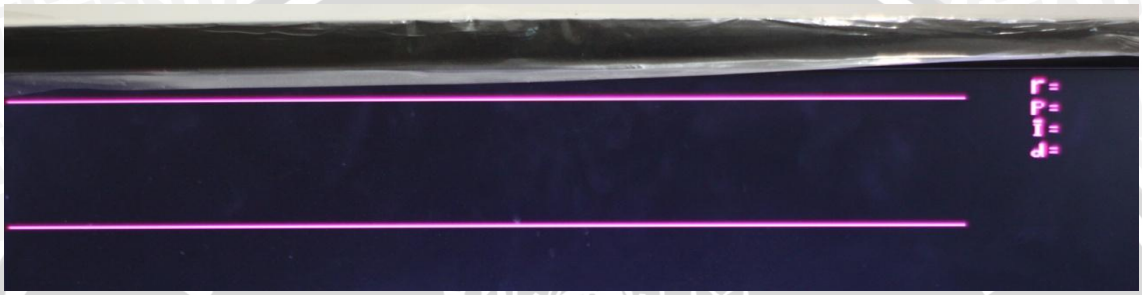
Gambar 1 Foto alat sistem pemonitoran



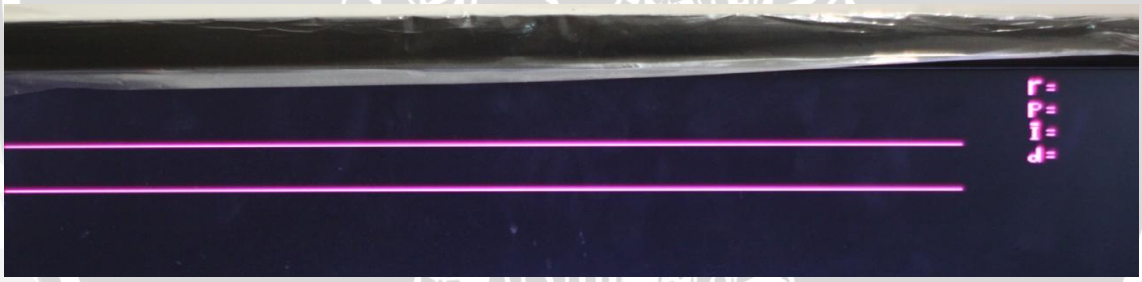
Gambar 2 Foto pengujian tampilan karakter *Keyboard* pada sistem pemonitoran



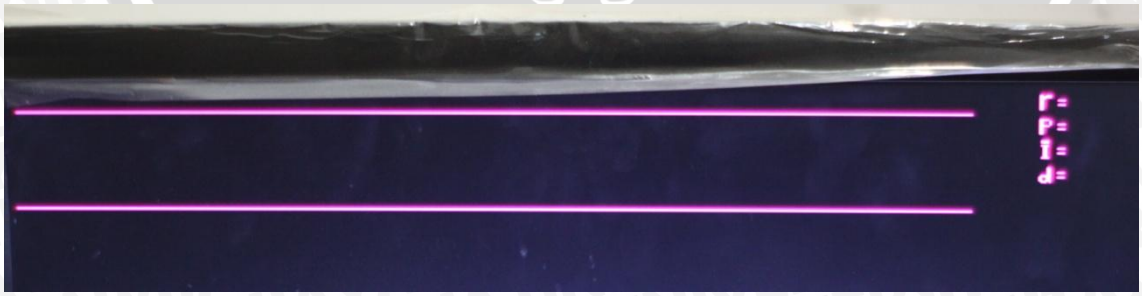
Gambar 3 Foto pengujian tampilan sinyal PWM "*LOW*" dan sinyal *Rotary* "*LOW*" pada sistem pemonitoran



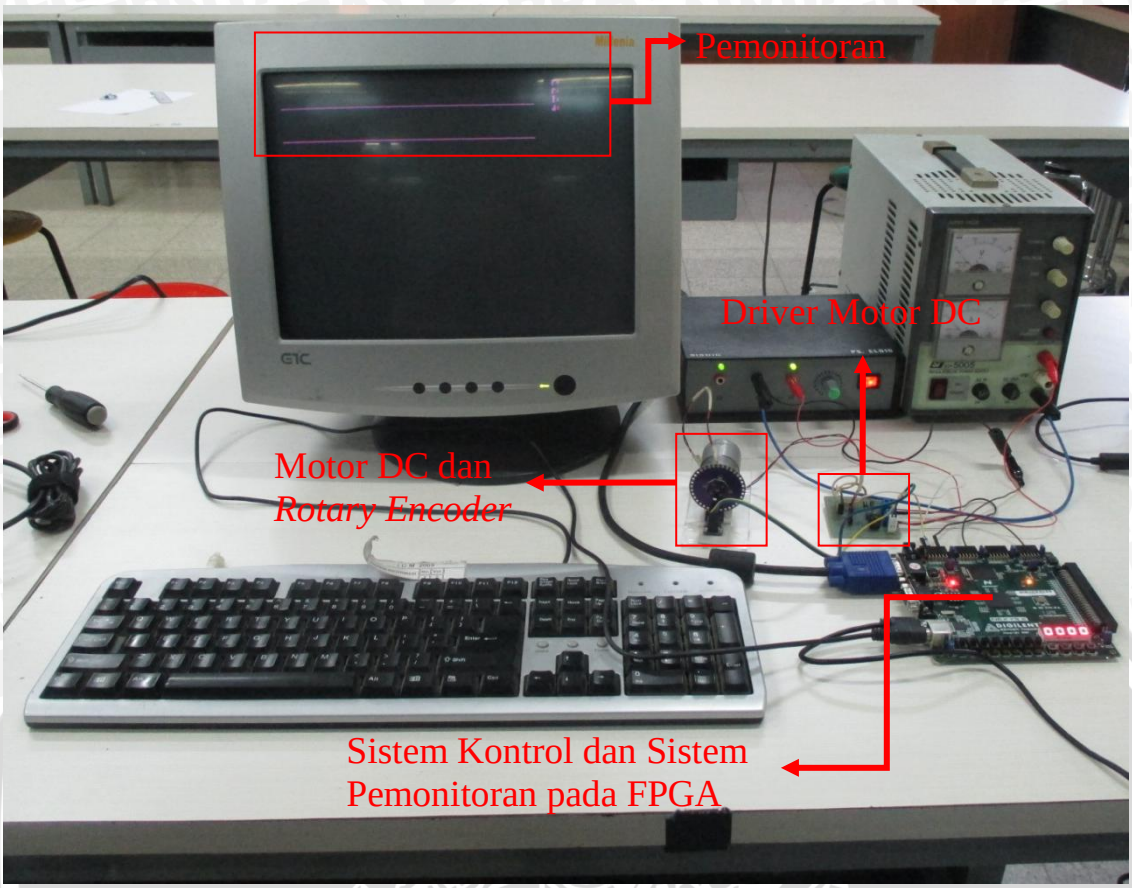
Gambar 4 Foto pengujian tampilan sinyal PWM "*HIGH*" dan sinyal *Rotary* "*LOW*" pada sistem pemonitoran



Gambar 5 Foto pengujian tampilan sinyal PWM "*LOW*" dan sinyal *Rotary* "*HIGH*" pada sistem pemonitoran



Gambar 6 Foto pengujian tampilan sinyal PWM "*HIGH*" dan sinyal *Rotary* "*HIGH*" pada sistem pemonitoran



Gambar 7 Foto alat sistem pemonitoran dan sistem kontrol Motor DC

Lampiran 2 VHDL *Main* Program Sistem Pemonitoran

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Main_Program_VRAM_Ref20 is
    Port (
pin_clk : in  STD_LOGIC;
pin_rst : in  STD_LOGIC;
PS2C : in  STD_LOGIC;
PS2D : in  STD_LOGIC;
PWM : in  STD_LOGIC;
ROT : in  STD_LOGIC;
port_code : out  STD_LOGIC_VECTOR (7
downto 0);
hs : out  STD_LOGIC;
vs : out  STD_LOGIC;
ro : out  STD_LOGIC_VECTOR (2 downto 0);
go : out  STD_LOGIC_VECTOR (2 downto 0);
bo : out  STD_LOGIC_VECTOR (2 downto 1));
end Main_Program_VRAM_Ref20;

architecture Behavioral of
Main_Program_VRAM_Ref20 is
signal clk, busy : STD_LOGIC;
signal make_code : STD_LOGIC_VECTOR (7
downto 0);
signal ascii : STD_LOGIC_VECTOR (7 downto
0);
signal x : STD_LOGIC_VECTOR (9 downto 0);
signal y : STD_LOGIC_VECTOR (9 downto 0);
-----------------------------------------
signal addrram : STD_LOGIC_VECTOR (16
downto 0);
signal enawk : STD_LOGIC;
signal enawt : STD_LOGIC;
signal enar : STD_LOGIC;
signal s : STD_LOGIC_VECTOR (2 downto 0);
-----------------------------------------
signal fontpix : STD_LOGIC_VECTOR (0 downto
0);
signal dop : STD_LOGIC_VECTOR (0 downto 0);
signal we : STD_LOGIC_VECTOR (0 downto 0);
signal sw : STD_LOGIC_VECTOR (0 downto 0);

COMPONENT gen_clk
        PORT(
                CLKIN_IN : IN std_logic;
                RST_IN : IN std_logic;
                CLKDV_OUT : OUT std_logic;
                CLKIN_IBUFG_OUT    :    OUT
std_logic;
                CLK0_OUT : OUT std_logic
                );
        END COMPONENT;

COMPONENT Keyboard_driver
        PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        ps2_clk : IN std_logic;
        ps2_dat : IN std_logic;
        busy : OUT std_logic;
        port_code : out  STD_LOGIC_VECTOR
        (7 downto 0);
        code : OUT std_logic_vector(7 downto 0)
                );
        END COMPONENT;

COMPONENT Keyboard_converter
        PORT(
        code : IN std_logic_vector(7 downto 0);
        ascii : OUT std_logic_vector(7 downto 0)
                );
        END COMPONENT;

COMPONENT Inst_scan_code
        PORT(
clk : IN std_logic;
rst : IN std_logic;
ascii : IN std_logic_vector(7 downto 0);
busy : IN std_logic;
x : IN std_logic_vector(9 downto 0);
y : IN std_logic_vector(9 downto 0);
enar : OUT std_logic;
we : OUT std_logic_vector(0 downto 0);
sw : OUT std_logic_vector(0 downto 0);
enawk : OUT std_logic;
enawt : OUT std_logic;
s : OUT std_logic_vector(2 downto 0);
addrram : OUT std_logic_vector(16 downto 0);
font_pix : OUT std_logic_vector(0 downto 0)
                );
        END COMPONENT;

COMPONENT Inst_VRAM
        PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        PWM : IN std_logic;
        ROT : IN std_logic;
        fontpix : IN std_logic_vector(0 downto 0);
        enawk : IN std_logic;
        enawt : IN std_logic;
        enar : IN std_logic;
        we : IN std_logic_vector(0 downto  0);
        sw : IN std_logic_vector(0 downto  0);
        s : IN std_logic_vector(2 downto 0);
addrram : IN std_logic_vector(16 downto 0);
        x : IN std_logic_vector(9 downto 0);
        y : IN std_logic_vector(9 downto 0);
        dop : OUT std_logic_vector(0 downto  0)
                );
        END COMPONENT;

COMPONENT Inst_VGA_Driver
        PORT(
clk : IN std_logic;
rst : IN std_logic;
```

```
ri : IN std_logic_vector(0 downto 0);
ro : OUT std_logic_vector(2 downto 0);
go : OUT std_logic_vector(2 downto 0);
bo : OUT std_logic_vector(2 downto 1);
hs : OUT std_logic;
vs : OUT std_logic;
x : OUT std_logic_vector(9 downto 0);
y : OUT std_logic_vector(9 downto 0)
        );
    END COMPONENT;

begin

Inst_gen_clk: gen_clk PORT MAP(
        CLKIN_IN => pin_clk,
        RST_IN => pin_rst,
        CLKDV_OUT => clk,
        CLKIN_IBUFG_OUT => open,
        CLK0_OUT => open
    );

Inst_Keyboard_driver: Keyboard_driver PORT
MAP(
        clk => clk,
        rst => pin_rst,
        ps2_clk => PS2C,
        ps2_dat => PS2D,
        busy => busy,
        port_code => port_code,
        code => make_code
    );

Inst_Keyboard_converter: Keyboard_converter
PORT MAP(
        code => make_code,
        ascii => ascii
    );

Inst_Inst_scan_code: Inst_scan_code PORT MAP(
        clk => clk,
        rst => pin_rst,
        ascii => ascii,
        busy => busy,
        x => x,
        y => y,
        enar => enar,
        we => we,
        sw => sw,
        enawk => enawk,
        enawt => enawt,
        s => s,
        addrram => addrram,
        font_pix => fontpix
    );


Inst_Inst_VRAM: Inst_VRAM PORT MAP(
        clk => clk,
        rst => pin_rst,
        PWM => PWM,
        ROT => ROT,
        fontpix => fontpix,
        enawk => enawk,
        enawt => enawt,
        enar => enar,
        we => we,
        sw => sw,
        s => s,
        addrram => addrram,
        x => x,
        y => y,
        dop => dop
    );

Inst_Inst_VGA_Driver: Inst_VGA_Driver PORT
MAP(
        clk => clk,
        rst => pin_rst,
        ri => dop,
        ro => ro,
        go => go,
        bo => bo,
        hs => hs,
        vs => vs,
        x => x,
        y => y
    );
end Behavioral;
```

Lampiran 3 VHDL *Keyboard Driver*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Keyboard_driver is
    Port ( clk : in  STD_LOGIC;
rst : in  STD_LOGIC;
ps2_clk : in  STD_LOGIC;
ps2_dat : in  STD_LOGIC;
busy : out STD_LOGIC;
port_code : out  STD_LOGIC_VECTOR (7
downto 0);
code : out  STD_LOGIC_VECTOR (7 downto 0));
end Keyboard_driver;

architecture Behavioral of Keyboard_driver is
-----------------------------------Signal Keyboard----
signal ps2_clk_syn : STD_LOGIC;
signal busy_P, busy_N : STD_LOGIC := '0';
signal cnt_P, cnt_N : unsigned (3 downto 0) :=
(others => '0');
-------------------------------------------------------------
----------Signal Register Data-------------------------
signal ps2_word_P, ps2_word_N : unsigned(10
downto 0) := (others => '0');       --lebar 11 bit ps2
data word --> start,D0-D7,parity,stop
signal code_P, code_N : unsigned (7 downto 0);
-------------------------------------------------------------
begin
process(clk, rst)
begin
            if rst = '1' then
            ps2_word_P <= (others => '0');
            busy_P <= '0';
            ps2_clk_syn <= '0';
            code_P <= (others => '0');
            cnt_P <= (others => '0');
            elsif clk'event and clk = '1' then
            ps2_clk_syn <= ps2_clk;
            code_P <= code_N;
            cnt_P <= cnt_N;
            busy_P <= busy_N;
            ps2_word_P <= ps2_word_N;
            end if;
end process;
-------------Register Input Keyboard--------------------
process(cnt_P, ps2_word_P, ps2_dat, ps2_clk_syn)
begin
if (ps2_clk_syn'event and ps2_clk_syn = '0') then  --falling edge of PS2 clock
cnt_N <= cnt_P + 1;
ps2_word_N(10 downto 0) <= ps2_dat &
ps2_word_P(10 downto 1);  --shift data in PS2 bit
end if;
if cnt_P = 11 then
cnt_N <= (others => '0');
end if;
end process;
----------------Proses Pembacaan Data---------------
process(cnt_P, busy_P, code_P, ps2_word_P)
begin
  busy_N <= busy_P;
  code_N <= code_P;
  if cnt_P = 0 then
            code_N <= code_P;
            busy_N <= '0';
  elsif cnt_P >= 1 and cnt_P <= 10 then
            code_N <= code_P;
            busy_N <= '1';
  else
            code_N <= ps2_word_P(8 downto 1);
            busy_N <= '0';
  end if;
end process;
----------------------Output-----------------------------
busy <= STD_LOGIC(busy_P);
code <= STD_LOGIC_VECTOR(code_P);
port_code <= STD_LOGIC_VECTOR(code_P);
-------------------------------------------------------------
end Behavioral;
```

Lampiran 4 VHDL *Keyboard Converter*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Keyboard_converter is
    Port ( code : in  STD_LOGIC_VECTOR (7
downto 0);
           ascii : out  STD_LOGIC_VECTOR (7
downto 0));
end Keyboard_converter;

architecture Behavioral of Keyboard_converter is

begin
process(code)
begin
        case code is
        when x"45" =>
                ascii <= x"30"; --0
        when x"16" =>
                ascii <= x"31"; --1
        when x"1E" =>
                ascii <= x"32"; --2
        when x"26" =>
                ascii <= x"33"; --3
        when x"25" =>
                ascii <= x"34"; --4
        when x"2E" =>
                ascii <= x"35"; --5
        when x"36" =>
                ascii <= x"36"; --6
        when x"3D" =>
                ascii <= x"37"; --7
        when x"3E" =>
                ascii <= x"38"; --8
        when x"46" =>
                ascii <= x"39"; --9
        when x"66" =>
ascii <= x"08"; --backspace (BS control code)
        when x"5A" =>
ascii <= x"0D"; --enter (CR control code)
        when x"75" =>
                ascii <= x"18"; --UP
        when x"72" =>
                ascii <= x"19"; --bottom
        when others =>
                ascii <= x"00";
        end case;
end process;
end Behavioral;
```

Lampiran 5 VHDL *Scan Code*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Inst_scan_code is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
ascii : in  STD_LOGIC_VECTOR (7 downto 0);
busy : in  STD_LOGIC;
x : in  STD_LOGIC_VECTOR (9 downto 0);
y : in  STD_LOGIC_VECTOR (9 downto 0);
enar : out STD_LOGIC;
we : out STD_LOGIC_VECTOR(0 downto 0);
sw : out STD_LOGIC_VECTOR(0 downto 0);
enawk : out STD_LOGIC;
enawt : out STD_LOGIC;
s : out STD_LOGIC_VECTOR(2 downto 0);
addrram : out  STD_LOGIC_VECTOR(16 downto
0);
font_pix : out  STD_LOGIC_VECTOR(0 downto
0));
end Inst_scan_code;

architecture Behavioral of Inst_scan_code is
signal addrk, addrt : STD_LOGIC_VECTOR (7
downto 0);--, addrt
signal addr_kbit, addr_tbit :
STD_LOGIC_VECTOR (2 downto 0);--, addr_tbit
signal k : std_logic_vector(2 downto 0);

COMPONENT Karakter
        PORT(
clk : IN std_logic;
rst : IN std_logic;
ascii : IN std_logic_vector(7 downto 0);
busy : IN std_logic;
xrom : IN std_logic_vector(9 downto 0);
yrom : IN std_logic_vector(9 downto 0);
we : OUT std_logic_vector(0 downto 0);
enawk : OUT std_logic;
enar : OUT std_logic;
sw : OUT std_logic_vector(0 downto 0);
addrram : OUT std_logic_vector(16 downto 0);
addrk_bit : OUT std_logic_vector(2 downto 0);
addr_k : OUT std_logic_vector(7 downto 0)
        );
        END COMPONENT;

COMPONENT Tampilan
        PORT(
x : IN std_logic_vector(9 downto 0);
y : IN std_logic_vector(9 downto 0);
addrt_bit : OUT std_logic_vector(2 downto 0);
addr_t : OUT std_logic_vector(7 downto 0);
enawt : OUT  STD_LOGIC;
s : OUT std_logic_vector(2 downto 0));
        END COMPONENT;
```

```
COMPONENT Scan_ROM
        PORT(
addrk_bit : IN std_logic_vector(2 downto 0);
addr_k : IN std_logic_vector(7 downto 0);
addrt_bit : IN std_logic_vector(2 downto 0);
addr_t : IN std_logic_vector(7 downto 0);
s : IN std_logic_vector(2 downto 0);
font_pix : OUT std_logic_vector(0 downto 0));
        END COMPONENT;
begin

Inst_Karakter: Karakter PORT MAP(
                clk => clk,
                rst => rst,
                ascii => ascii,
                busy => busy,
                xrom => x,
                yrom => y,
                we => we,
                enawk => enawk,
                enar => enar,
                sw => sw,
                addrram => addrram,
                addrk_bit => addr_kbit,
                addr_k => addrk
        );

Inst_Tampilan: Tampilan PORT MAP(
                x => x,
                y => y,
                addrt_bit => addr_tbit,
                addr_t => addrt,
                enawt => enawt,
                s => k
        );

s <= k;-- when k = 0 else enawk;

Inst_Scan_ROM: Scan_ROM PORT MAP(
                addrk_bit => addr_kbit,
                addr_k => addrk,
                addrt_bit => addr_tbit,
                addr_t => addrt,
                s => k,
                font_pix => font_pix
        );
end Behavioral;
```

Lampiran 6 VHDL Karakter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Karakter is
   Port ( clk : in  STD_LOGIC;
rst : in  STD_LOGIC;
ascii : in  STD_LOGIC_VECTOR (7 downto 0);
busy : in  STD_LOGIC;
xrom : in  STD_LOGIC_VECTOR (9 downto 0);
yrom : in  STD_LOGIC_VECTOR (9 downto 0);
we : out STD_LOGIC_VECTOR (0 downto 0);
enawk : out STD_LOGIC;
enar : out STD_LOGIC;
sw : out STD_LOGIC_VECTOR (0 downto 0);
addrram : out STD_LOGIC_VECTOR (16 downto
0);
addrk_bit : out  STD_LOGIC_VECTOR (2 downto
0);
addr_k : out  STD_LOGIC_VECTOR (7 downto
0));
end Karakter;

architecture Behavioral of Karakter is
signal xrp, xrn : std_logic_vector(3 downto 0) :=
(others => '0'); --alamat rom char
signal hp, hn : std_logic_vector(3 downto 0) :=
(others => '0'); --hurup
signal bp, bn : std_logic_vector(2 downto 0) :=
(others => '0'); --baris
signal busy_p : std_logic;
signal addrk_char : std_logic_vector(3 downto 0);
signal addrk_row  : std_logic_vector(3 downto 0);
signal wep, wen : unsigned(0 downto 0) := (others
=> '1');
signal enawp, enawn : std_logic;

begin

addrk_row <= yrom(3 downto 0);
process(clk, rst)
begin
if rst = '1' then
        hp <= (others => '0');
        bp <= (others => '0');
        busy_p <= '0';
        wep <= (others => '0');
        xrp <= (others => '0');
elsif clk'event and clk = '0' then
        hp <= hn;
        bp <= bn;
        xrp <= xrn;
        wep <= wen;
        enawp <= enawn;
        busy_p <= busy;
end if;
end process;
```

```
process(busy_p, hp, bp, ascii, xrp)
begin
if(busy_p'event and busy_p = '0') then
case (ascii) is
when "00110000" => xrn <= x"5";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00110001" => xrn <= x"6";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;

when "00110010" => xrn <= x"7";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00110011" => xrn <= x"8";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00110100" => xrn <= x"9";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00110101" => xrn <= x"A";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00110110" => xrn <= x"B";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00110111" => xrn <= x"C";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00111000" => xrn <= x"D";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00111001" => xrn <= x"E";
if hp = 10 then
hn <= hp;
else
hn <= hp + 1;
end if;
when "00001101" => xrn <= x"0";
if hp < 10 then
hn <= "1010";
else
hn <= hp; --enter
end if;
when "00001000" => xrn <= x"0";
if hp = 0 then
hn <= hp;
else
hn <= hp - 1; --backspace
end if;
when "00011000" => --UP
hn <= (others => '0');
if bp = 0 then
bn <= bp;
else
bn <= bp - 1;
end if;
when "00011001" => --DOWN
hn <= (others => '0');
if bp = 6 then
 bn <= bp;
else
bn <= bp + 1;
when others => xrn <= xrp; bn <= bp; hn <= hp;
end case;
end if;
end process;

process(xrom, yrom, xrp, hp, bp, enawp)
begin
enawn <= enawp;
case (bp) is
when "000" =>
------------------baris 1 ---------------------------------
            case (hp) is
            when "0010" =>
            if xrom >= 528 and xrom <= 535 and
yrom >= 0 and yrom <= 15 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "0100" =>
            if xrom >= 536 and xrom <= 543 and
yrom >= 0 and yrom <= 15 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
```

```
            when "0110" =>
            if xrom >= 544 and xrom <= 551 and
yrom >= 0 and yrom <= 15 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1000" =>
            if xrom >= 552 and xrom <= 559 and
yrom >= 0 and yrom <= 15 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1010" =>
            if xrom >= 560 and xrom <= 567 and
yrom >= 0 and yrom <= 15 then
            addrk_char <= x"0";
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when others =>
            addrk_char <= xrp;
            enawn <= '0';
            end case;
            ------------------------------------------------
            when "010" =>
            ------------------baris 2 ----------------------
            case (hp) is
            when "0010" =>
            if xrom >= 528 and xrom <= 535 and
yrom >= 16 and yrom <= 31 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "0100" =>
            if xrom >= 536 and xrom <= 543 and
yrom >= 16 and yrom <= 31 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "0110" =>
            if xrom >= 544 and xrom <= 551 and
yrom >= 16 and yrom <= 31 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1000" =>
            if xrom >= 552 and xrom <= 559 and
yrom >= 16 and yrom <= 31 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1010" =>
            if xrom >= 560 and xrom <= 567 and
yrom >= 16 and yrom <= 31 then
            addrk_char <= x"0";
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when others =>
            addrk_char <= (others => '0');
            enawn <= '0';
            end case;
            ------------------------------------------------
            when "100" =>
            ------------------baris 3 ----------------------
case (hp) is
            when "0010" =>
            if xrom >= 528 and xrom <= 535 and
yrom >= 32 and yrom <= 47 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "0100" =>
            if xrom >= 536 and xrom <= 543 and
yrom >= 32 and yrom <= 47 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "0110" =>
            if xrom >= 544 and xrom <= 551 and
yrom >= 32 and yrom <= 47 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1000" =>
            if xrom >= 552 and xrom <= 559 and
yrom >= 32 and yrom <= 47 then
            addrk_char <= xrp;
            enawn <= '1';
```

```
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1010" =>
            if xrom >= 560 and xrom <= 567 and
yrom >= 32 and yrom <= 47 then
            addrk_char <= x"0";
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when others =>
            addrk_char <= (others => '0');
            enawn <= '0';
            end case;
            ------------------------------------------------
            when "110" =>
            ------------------baris 4 ----------------------
            case (hp) is
            when "0010" =>
            if xrom >= 528 and xrom <= 535 and
yrom >= 48 and yrom <= 63 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "0100" =>
            if xrom >= 536 and xrom <= 543 and
yrom >= 48 and yrom <= 63 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "0110" =>
            if xrom >= 544 and xrom <= 551 and
yrom >= 48 and yrom <= 63 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1000" =>
            if xrom >= 552 and xrom <= 559 and
yrom >= 48 and yrom <= 63 then
            addrk_char <= xrp;
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when "1010" =>
            if xrom >= 560 and xrom <= 567 and
yrom >= 32 and yrom <= 47 then
            addrk_char <= x"0";
            enawn <= '1';
            else
            addrk_char <= (others => '0');
            enawn <= '0';
            end if;
            when others =>
            addrk_char <= (others => '0');
            enawn <= '0';
            end case;
            ------------------------------------------------
            when others =>
            addrk_char <= (others => '0');
            enawn <= '0';
            end case;
            end process;

process(xrom, yrom, wep)
begin
wen <= wep;
if xrom = 799 and yrom = 524 then
wen <= not wep;
else
wen <= wep;
end if;
end process;

process(yrom, wep)
begin
if wep = 0 then
    if yrom >= 128 then
    enar <= '0';
    else
    enar <= '1';
    end if;
else
enar <= '0';
end if;
end process;
    ----------------------------OUTPUT---------------------
addr_k <= addrk_char & addrk_row;
addrk_bit <= xrom(2 downto 0);
we <= STD_LOGIC_VECTOR(wep);
sw <= STD_LOGIC_VECTOR(wep);
enawk <= enawp;
addrram <= yrom(6 downto 0)&(xrom);
------------------------------------------------------------
end Behavioral;
```

Lampiran 7 VHDL Tampilan

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Tampilan is
    Port ( x : in STD_LOGIC_VECTOR (9 downto 0);
y : in STD_LOGIC_VECTOR (9 downto 0);
addrt_bit : out STD_LOGIC_VECTOR (2 downto 0);
addr_t : out STD_LOGIC_VECTOR (7 downto 0);
enawt : out STD_LOGIC;
s : out STD_LOGIC_VECTOR (2 downto 0));
end Tampilan;

architecture Behavioral of Tampilan is
signal addrt_char : std_logic_vector(3 downto 0);
signal addrt_row : std_logic_vector(3 downto 0);
signal enaw : std_logic;
begin

addrt_row <= (y (3 downto 0));


process(x, y)
begin
if y >= 0 and y <= 15 then
if x >= 512 and x <= 519 then
        addrt_char <=   "0001";
        enaw <= '1';
elsif x >= 520 and x <= 527 then
        addrt_char <=   "1111";
        enaw <= '1';
else
        addrt_char <=   "0000";
        enaw <= '0';
end if;
elsif y >= 16 and y <= 31 then
        if x >= 512 and x <= 519 then
                addrt_char <=   "0010";
                enaw <= '1';
                elsif x >= 520 and x <= 527 then
                addrt_char <=   "1111";
                enaw <= '1';
        else
                addrt_char <=   "0000";
                enaw <= '0';
        end if;
elsif y >= 32 and y <= 47 then
        if x >= 512 and x <= 519 then
                addrt_char <=   "0011";
                enaw <= '1';
        elsif x >= 520 and x <= 527 then
                addrt_char <=   "1111";
                enaw <= '1';
        else
                addrt_char <=   "0000";
                enaw <= '0';
        end if;
elsif y >= 48 and y <= 63 then
        if x >= 512 and x <= 519 then
                addrt_char <=   "0100";
                enaw <= '1';
        elsif x >= 520 and x <= 527 then
                addrt_char <=   "1111";
                enaw <= '1';
        else
                addrt_char <=   "0000";
                enaw <= '0';
        end if;
else
        addrt_char <=   "0000";
        enaw <= '0';
end if;
end process;

process(x, y)
begin
if y <= 63 and x >= 512 and x <= 527 then
        s <= "000"; --tampilan
elsif y <= 63 and x >= 528 and x <= 599 then
        s <= "001";
elsif y <= 63 and x >= 0 and x <= 479 then
        s <= "010"; --pwm
elsif y >= 64 and y <= 127 and x >= 0 and x <= 479
then
        s <= "011"; --rot
elsif y <= 63 and x >= 480 and x <= 511 then
        <= "100"; --No display
elsif y >= 64 and y <= 127 and x >= 480 and x <=
639 then
        s <= "101"; --No display
else
        s <= "110"; --No display
end if;
end process;
---------OUTPUT-----------------------------------------
        enawt <= enaw;
        addrt_bit <= (x (2 downto 0));
        addr_t <= addrt_char & addrt_row;
        ------------------------------------------------
---------------------------------
end Behavioral;
```

Lampiran 8 VHDL *Scan* ROM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Scan_ROM is
    Port ( addrk_bit : in  STD_LOGIC_VECTOR (2
downto 0);
addr_k : in  STD_LOGIC_VECTOR (7 downto 0);
addrt_bit : in  STD_LOGIC_VECTOR (2 downto
0);
addr_t : in  STD_LOGIC_VECTOR (7 downto 0);
s : in std_logic_vector(2 downto 0);
font_pix : out  STD_LOGIC_VECTOR (0 downto
0));
end Scan_ROM;

architecture Behavioral of Scan_ROM is
signal addr : std_logic_vector(7 downto 0);
signal dout : std_logic_vector(7 downto 0);
signal addr_bit : std_logic_vector(2 downto 0);
signal word_font : std_logic_vector(7 downto 0); --
data
signal bit_pix : std_logic;

constant width_addr : integer  := 8;
constant width_dout : integer  := 8;
type rom_font is array (0 to 2**width_addr-1) of
STD_LOGIC_VECTOR (width_dout-1 downto 0);
--2^8 by 8 bit = 2048 bit
constant ROM : rom_font :=
        (       "00000000", --00 x00
                "00000000", --01
                "00000000", --02null
                "00000000", --03
                "00000000", --04
                "00000000", --05
                "00000000", --06
                "00000000", --07
                "00000000", --08
                "00000000", --09
                "00000000", --10
                "00000000", --11-
                "00000000", --12
                "00000000", --13
                "00000000", --14
                "00000000", --15--
                "00000000", --16 x01
                "00000000", --17
                "01101110", --18r
                "01111110", --19
                "01110000", --20
                "01100000", --21
                "01100000", --22
                "01100000", --23
                "01100000", --24
                "01100000", --25
                "01100000", --26
                "01100000", --27-
                "00000000", --28
                "00000000", --29
                "00000000", --30
                "00000000", --31--
                "00000000", --32 x02
                "00000000", --33
                "01111100", --34p
                "01100010", --35
                "01100010", --36
                "01100010", --37
                "01111100", --38
                "01100000", --39
                "01100000", --40
                "01100000", --41
                "01100000", --42
                "01100000", --43-
                "00000000", --44
                "00000000", --45
                "00000000", --46
                "00000000", --47--
                "00111100", --48i x03
                "00111100", --49
                "00000000", --50
                "00111100", --51
                "00011000", --52
                "00011000", --53
                "00011000", --54
                "00011000", --55
                "00011000", --56
                "00011000", --57
                "00011000", --58
                "00111100", --59-
                "00000000", --60
                "00000000", --61
                "00000000", --62
                "00000000", --63--
                "00000000", --64 x04
                "00000000", --65
                "00000110", --66d
                "00000110", --67
                "00000110", --68
                "00000110", --69
                "00000110", --70
                "00111110", --71
                "01101110", --72
                "01000110", --73
                "01101110", --74
                "00111010", --75-
                "00000000", --76
                "00000000", --77
                "00000000", --78
                "00000000", --79--
                "00000000", --80 x05
                "00000000", --81
                "00111100", --82 0
                "01100110", --83
                "01000010", --84
                "01000010", --85
                "01000010", --86
```

"01000010", --87
"01000010", --88
"01000010", --89
"01100110", --90
"00111100", --91-
"00000000", --92
"00000000", --93
"00000000", --94
"00000000", --95--
"00000000", --96 x06
"00000000", --97
"00111000", --98 1
"01111000", --99
"11011000", --100
"00011000", --101
"00011000", --102
"00011000", --103
"00011000", --104
"00011000", --105
"00011000", --106
"01111110", --107-
"00000000", --108
"00000000", --109
"00000000", --110
"00000000", --111--
"00000000", --112 x07
"00000000", --113
"00111100", --114 2
"01000010", --115
"01000001", --116
"00000001", --117
"00000010", --118
"00001100", --119
"00011000", --120
"00110000", --121
"01100000", --122
"01111111", --123-
"00000000", --124
"00000000", --125
"00000000", --126
"00000000", --127--
"00000000", --128 x08
"00000000", --129
"00111100", --130 3
"01000010", --131
"01000001", --132
"00000001", --133
"00111110", --134
"00000010", --135
"00000001", --136
"01000001", --137
"01000010", --138
"00111100", --139-
"00000000", --140
"00000000", --141
"00000000", --142
"00000000", --143--
"00000000", --144 x09
"00000000", --145
"00000110", --146 4

"00001010", --147
"00010010", --148
"00100010", --149
"01000010", --150
"01111111", --151
"00000010", --152
"00000010", --153
"00000010", --154
"00000111", --155-
"00000000", --156
"00000000", --157
"00000000", --158
"00000000", --159--
"00000000", --160 x0A
"00000000", --161
"01111110", --162 5
"01000000", --163
"01000000", --164
"01111100", --165
"00000010", --166
"00000010", --167
"00000010", --168
"01000010", --169
"01000010", --170
"00111100", --171-
"00000000", --172
"00000000", --173
"00000000", --174
"00000000", --175--
"00000000", --176 x0B
"00000000", --177
"00011110", --178 6
"00100000", --179
"01000000", --180
"01000000", --181
"01011100", --182
"01100110", --183
"01000010", --184
"01100110", --185
"00111100", --186
"00000000", --187-
"00000000", --188
"00000000", --189
"00000000", --190
"00000000", --191--
"00000000", --192 x0C
"00000000", --193
"01111110", --194 7
"01000010", --195
"00000010", --196
"00000100", --197
"00001000", --198
"00010000", --199
"00010000", --200
"00010000", --201
"00010000", --202
"00010000", --203-
"00000000", --204
"00000000", --205
"00000000", --206

```
"00000000", --207--
"00000000", --208 x0D
"00000000", --209
"00111100", --210 8
"01100110", --211
"01000010", --212
"01000010", --213
"00111100", --214
"01000010", --215
"01000010", --216
"01000010", --217
"01100110", --218
"00111100", --219-
"00000000", --220
"00000000", --221
"00000000", --222
"00000000", --223--
"00000000", --224 x0E
"00000000", --225
"00011000", --226 9
"00100100", --227
"01000010", --228
"01000010", --229
"00100110", --230
"00011010", --231
"00000010", --232
"01000010", --233
"01000010", --234
"00111100", --235-
"00000000", --236
"00000000", --237
"00000000", --238
"00000000", --239--
"00000000", --240 x0F
"00000000", --241
"00000000", --242
"00000000", --243
"00000000", --244
"00111100", --245
"00000000", --246
"00000000", --247
"00111100", --248
"00000000", --249
"00000000", --250
"00000000", --251-
"00000000", --252
"00000000", --253
"00000000", --254
"00000000"); --255--

begin
---MUX---
process(s, addr_k, addr_t, addrk_bit, addrt_bit)
        begin
                if s = 0 then
                        addr <=  addr_t;
                        addr_bit <= addrt_bit;
                elsif s = 1 then
                        addr <=  addr_k;
                        addr_bit <= addrk_bit;
                else
                        addr <=  x"00";
                        addr_bit <= "000";
                end if;
        end process;
---------
---ROM---
dout <= ROM(to_integer(unsigned(addr)));
word_font <= dout;
---------
---MUX 3 bit---
bit_pix     <=     word_font(to_integer(unsigned(not
addr_bit)));
---------------
font_pix(0) <= bit_pix;
---------------
end Behavioral;
```

Lampiran 9 VHDL *Scan* PWM dan *Rotary*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Scan_PWM_Rotary is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           sinyal_pwm : in  STD_LOGIC;
           sinyal_Rotary : in  STD_LOGIC;
           puls_pwm : out  STD_LOGIC;
           puls_Rotary : out  STD_LOGIC);
end Scan_PWM_Rotary;

architecture Behavioral of Scan_PWM_Rotary is
COMPONENT Scan_PWM
        PORT(
                clk : IN std_logic;
                rst : IN std_logic;
                sinyal_pwm : IN std_logic;
                puls_pwm : OUT std_logic
                );
        END COMPONENT;

COMPONENT Scan_Rotary
        PORT(
                clk : IN std_logic;
                rst : IN std_logic;
                sinyal_Rotary : IN std_logic;
                puls_Rotary : OUT std_logic
                );
        END COMPONENT;
begin
Inst_Scan_PWM: Scan_PWM PORT MAP(
                clk => clk,
                rst => rst,
                sinyal_pwm => sinyal_pwm,
                puls_pwm => puls_pwm
        );

Inst_Scan_Rotary: Scan_Rotary PORT MAP(
                clk => clk,
                rst => rst,
                sinyal_Rotary => sinyal_Rotary,
                puls_Rotary => puls_Rotary
        );
end Behavioral;
```

Lampiran 10 VHDL *Scan* PWM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Scan_PWM is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           sinyal_pwm : in  STD_LOGIC;
           puls_pwm : out  STD_LOGIC);
end Scan_PWM;

architecture Behavioral of Scan_PWM is
signal pwm_NS, pwm_PS : std_logic;
signal puls_NS, puls_PS : std_logic :=  '0';
signal  state_n,  state_p  :  unsigned(1  downto  0)  :=
(others => '0');
signal state1_n, state1_p : std_logic :=  '0';
signal cnt_NS, cnt_PS : unsigned(14 downto 0) :=
(others => '0');--scan sinyal low
signal cnt1_NS, cnt1_PS : unsigned(14 downto 0)
:= (others => '0');--scan sinyal high
signal  r_NS,  r_PS  :  unsigned(14  downto  0)  :=
(others => '0');--batas nilai scan sinyal low
signal  r1_NS,  r1_PS  :  unsigned(14  downto  0)  :=
(others => '0');--batas nilai scan sinyal high
signal  a  :  integer  range  0  to  32766  :=  0;--(15
downto 0) := (others => '0');-- nilai scan sinyal
integer
signal  b  :  integer  range  0  to  32766  :=  0;--(15
downto 0) := (others => '0');-- nilai scan sinyal
integer
signal c : integer range 0 to 31 := 0; --unsigned(15
downto 0) := (others => '0');--batas nilai scan sinyal
low per 1000
signal d : integer range 0 to 31 := 0;--(15 downto 0)
:= (others => '0');--batas nilai scan sinyal high per
1000
signal e : integer range 0 to 31 := 0;--(15 downto 0)
:=  (others  =>  '0');--batas  nilai  scan  sinyal  dengan
nilai a atau b
signal cnt2_NS, cnt2_PS : integer range 0 to 31 :=
0; -- scan sinyal sesuai batas nilai scan sinyal utama
begin
--memory
process (clk,rst)
begin
if rst = '1' then
            pwm_PS <= '0';
            puls_PS <= '0';
            state_p <= (others => '0');
            state1_p <= '0';
            cnt_PS <= (others => '0');
            cnt1_PS <= (others => '0');
            r_PS <= (others => '0');
            r1_PS <= (others => '0');
            cnt2_PS <= 0;
elsif (clk'event and clk = '1') then
```

```
            pwm_PS <= pwm_NS;
            puls_PS <= puls_NS;
            state_p <= state_n;
            state1_p <= state1_n;
            cnt_PS <= cnt_NS;
            cnt1_PS <= cnt1_NS;
            r_PS <= r_NS;
            r1_PS <= r1_NS;
            cnt2_PS <= cnt2_NS;
end if;
end process;

process(state_p, state1_p, cnt_PS, cnt1_PS,
sinyal_pwm, pwm_PS)
begin
            state_n <= state_p;
            state1_n <= state1_p;
            pwm_NS <= sinyal_pwm;
            cnt_NS <= cnt_PS;
            cnt1_NS <= cnt1_PS;
if sinyal_pwm = '0' then
            cnt_NS <= cnt_PS + 1;  --memory
            cnt1_NS <= cnt1_PS;
            state1_n <= '0';
else
            cnt_NS <= cnt_PS;
            cnt1_NS <= cnt1_PS + 1;  --memory
            state1_n <= '1';
end if;
if pwm_PS = '0' and sinyal_pwm = '1' then
            state_n <= "01";
            cnt_NS <= cnt_PS;
            cnt1_NS <= (others => '0');
elsif pwm_PS = '1' and sinyal_pwm = '0' then
            cnt_NS <= (others => '0');
            cnt1_NS <= cnt1_PS;
            state_n <= "10";
else
            state_n <= (others => '0');
end if;
end process;
process(state_p,  state1_p,  cnt_PS,  cnt1_PS,  r_PS,
r1_PS)
begin
r_NS <= r_PS;
r1_NS <= r1_PS;
if state1_p = '0' then
if cnt_PS = 25000 then
r_NS <= "110000110101000";
r1_NS <= (others => '0');
else
r_NS <= r_PS;
r1_NS <= r1_PS;
end if;
else
if cnt1_PS = 25000 then
r_NS <= (others => '0');
r1_NS <= "110000110101000";
else
r_NS <= r_PS;
r1_NS <= r1_PS;
end if;
end if;
if state_p = 1 then
            r_NS <= cnt_PS;
elsif state_p = 2 then
            r1_NS <= cnt1_PS;
end if;
end process;

process(r_PS, r1_PS)
begin
a <= TO_INTEGER (r_PS); --convert ke integer
b <= TO_INTEGER (r1_PS); --convert ke integer
end process;

process(a, b)
begin
c <= a/(2*2*2*2*2*2*2*2*2*2);--per 1000;
d <= b/(2*2*2*2*2*2*2*2*2*2);--per 1000;
end process;

e <= c + d;
process(cnt2_PS, e)
begin
cnt2_NS <= cnt2_PS + 1;
if (cnt2_PS = e) then
cnt2_NS <= 0;
-------------------------------------
else
if cnt2_PS = 31 then
            cnt2_NS <= 0;
end if;
end if;
end process;

process(cnt2_PS, c, d, puls_PS)
begin
            puls_NS <= puls_PS;
if c > 0 then
            if cnt2_PS <= c then
                        puls_NS <= '0';
            else
                        puls_NS <= '1';
            end if;
else
            if cnt2_PS <= d then
                        puls_NS <= '1';
            else
                        puls_NS <= '0';
            end if;
end if;
end process;
            --Komb2
            puls_pwm <= (puls_PS);
end Behavioral;
```

Lampiran 11 VHDL *Scan Rotary*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Scan_Rotary is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           sinyal_Rotary : in  STD_LOGIC;
           puls_Rotary : out  STD_LOGIC);
end Scan_Rotary;

architecture Behavioral of Scan_Rotary is
signal Rotary_NS, Rotary_PS : std_logic;
signal puls_NS, puls_PS : std_logic :=  '0';
signal  state_n,  state_p  :  unsigned(1  downto  0)  :=
(others => '0');
signal state1_n, state1_p : std_logic :=  '0';
signal cnt_NS, cnt_PS : unsigned(17 downto 0) :=
(others => '0');--scan sinyal low
signal cnt1_NS, cnt1_PS : unsigned(17 downto 0)
:= (others => '0');--scan sinyal high
signal  r_NS,  r_PS  :  unsigned(17  downto  0)  :=
(others => '0');--batas nilai scan sinyal low
signal  r1_NS,  r1_PS  :  unsigned(17  downto  0)  :=
(others => '0');--batas nilai scan sinyal high
signal  a  :  integer  range  0  to  262142  :=  0;--batas
nilai scan sinyal low per 1000
signal  b  :  integer  range  0  to  262142  :=  0;--batas
nilai scan sinyal high per 1000
signal c : integer range 0 to 255 := 0; --batas nilai
scan sinyal dengan nilai a atau b
signal  d  :  integer  range  0  to  255  :=  0;--batas  nilai
scan sinyal low per 1000
signal  e  :  integer  range  0  to  255  :=  0;--batas  nilai
scan sinyal high per 1000
signal cnt2_NS, cnt2_PS : integer range 0 to 255 :=
0; -- scan sinyal sesuai batas nilai scan sinyal utama
begin
--memory
process (clk,rst)
begin
if rst = '1' then
        Rotary_PS <= '0';
        puls_PS <= '0';
        state_p <= (others => '0');
        state1_p <= '0';
        cnt_PS <= (others => '0');
        cnt1_PS <= (others => '0');
        r_PS <= (others => '0');
        r1_PS <= (others => '0');
        cnt2_PS <= 0;
elsif (clk'event and clk = '1') then
        Rotary_PS <= Rotary_NS;
        puls_PS <= puls_NS;
        state_p <= state_n;
        state1_p <= state1_n;
        cnt_PS <= cnt_NS;
        cnt1_PS <= cnt1_NS;
        r_PS <= r_NS;
        r1_PS <= r1_NS;
        cnt2_PS <= cnt2_NS;
end if;
end process;
process(state_p,  state1_p,  cnt_PS,  cnt1_PS,
sinyal_Rotary, Rotary_PS)
begin
        state_n <= state_p;
        state1_n <= state1_p;
        Rotary_NS <= sinyal_Rotary;
        cnt_NS <= cnt_PS;
        cnt1_NS <= cnt1_PS;

if sinyal_Rotary = '0' then
        cnt_NS <= cnt_PS + 1;  --memory
        cnt1_NS <= cnt1_PS;
        state1_n <= '0';
else
        cnt_NS <= cnt_PS;
        cnt1_NS <= cnt1_PS + 1;  --memory
        state1_n <= '1';

end if;

if Rotary_PS = '0' and sinyal_Rotary = '1' then
        state_n <= "01";
        cnt_NS <= cnt_PS;
        cnt1_NS <= (others => '0');
elsif Rotary_PS = '1' and sinyal_Rotary = '0' then
        cnt_NS <= (others => '0');
        cnt1_NS <= cnt1_PS;
        state_n <= "10";
else
        state_n <= (others => '0');
end if;
end process;

process(state_p,  state1_p,  cnt_PS,  cnt1_PS,  r_PS,
r1_PS)
begin
r_NS <= r_PS;
r1_NS <= r1_PS;
if state1_p = '0' then
    if cnt_PS = 250000 then
        r_NS <= "111101000010010000";
        r1_NS <= (others => '0');
    else
        r_NS <= r_PS;
        r1_NS <= r1_PS;
    end if;
else
    if cnt1_PS = 250000 then
        r_NS <= (others => '0');
        r1_NS <= "111101000010010000";
    else
        r_NS <= r_PS;
        r1_NS <= r1_PS;
    end if;
end if;
```

```
if state_p = 1 then
        r_NS <= cnt_PS;
elsif state_p = 2 then
        r1_NS <= cnt1_PS;
end if;
end process;

process(r_PS, r1_PS)
begin
--default :
a <= TO_INTEGER (r_PS); --convert ke integer
b <= TO_INTEGER (r1_PS); --convert ke integer
end process;

process(a, b)
begin
c <= a/(2*2*2*2*2*2*2*2*2*2);--per 1000;
d <= b/(2*2*2*2*2*2*2*2*2*2);--per 1000;
end process;

e <= c + d;

process(cnt2_PS, e)
begin
--default :
cnt2_NS <= cnt2_PS + 1;
if (cnt2_PS = e) then
        cnt2_NS <= 0;
-------------------------------------
else
if cnt2_PS = 255 then
        cnt2_NS <= 0;
end if;
end if;
end process;

process(cnt2_PS, c, d, puls_PS)
begin
puls_NS <= puls_PS;
if c > 0 then
if cnt2_PS <= c then
        puls_NS <= '0';
else
        puls_NS <= '1';
end if;
else
if cnt2_PS <= d then
        puls_NS <= '1';
else
        puls_NS <= '0';
end if;
end if;
end process;
--Komb2
puls_Rotary <= (puls_PS);
end Behavioral;
```

Lampiran 12 VHDL VRAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_unsigned.all;

entity Inst_VRAM is
    Port ( clk : in  STD_LOGIC;
rst : in  STD_LOGIC;
PWM : in  STD_LOGIC;
ROT : in  STD_LOGIC;
fontpix : in STD_LOGIC_VECTOR (0 downto 0);
enawk : in STD_LOGIC;
enawt : in STD_LOGIC;
enar : in STD_LOGIC;
we : in STD_LOGIC_VECTOR (0 downto 0);
sw : in STD_LOGIC_VECTOR (0 downto 0);
s : in STD_LOGIC_VECTOR (2 downto 0);
addrram : in STD_LOGIC_VECTOR (16 downto
0);
x : in STD_LOGIC_VECTOR (9 downto 0);
y : in STD_LOGIC_VECTOR (9 downto 0);
dop : out STD_LOGIC_VECTOR (0 downto 0));
end Inst_VRAM;

architecture Behavioral of Inst_VRAM is
-----------------------------------------------
signal datram : STD_LOGIC_VECTOR (0 downto
0);
signal datpr : STD_LOGIC_VECTOR (0 downto
0);
signal enawpr : STD_LOGIC;
signal enaw : STD_LOGIC;

COMPONENT graphic
        PORT(
clk : IN std_logic;
rst : IN std_logic;
pwm : IN std_logic;
rot : IN std_logic;
x : IN std_logic_vector(9 downto 0);
y : IN std_logic_vector(9 downto 0);
dat : OUT std_logic_vector(0 downto 0);
enaw : OUT std_logic
                );
        END COMPONENT;
COMPONENT MUX
        PORT(
s : IN std_logic_vector(2 downto 0);
fontpix : IN std_logic_vector(0 downto 0);
dat : IN std_logic_vector(0 downto 0);
enawpr : IN std_logic;
enawt : IN std_logic;
enawk : IN std_logic;
enaw : OUT std_logic;
dip : OUT std_logic_vector(0 downto 0)
                );
        END COMPONENT;
```

```
COMPONENT G_RAM
        PORT(
clk : IN std_logic;
rst : IN std_logic;
we : IN std_logic_vector(0 downto 0);
enaw : IN std_logic;
enar : IN std_logic;
sw : IN std_logic_vector(0 downto 0);
dip : IN std_logic_vector(0 downto 0);
addrram : IN std_logic_vector(16 downto 0);
dop : OUT std_logic_vector(0 downto 0)
                );
        END COMPONENT;

begin
Inst_graphic: graphic PORT MAP(
                clk => clk,
                rst => rst,
                pwm => PWM,
                rot => ROT,
                x => x,
                y => y,
                dat => datpr,
                enaw => enawpr
        );
Inst_MUX: MUX PORT MAP(
                s => s,
                fontpix => fontpix,
                dat => datpr,
                enawpr => enawpr,
                enawt => enawt,
                enawk => enawk,
                enaw => enaw,
                dip => datram
        );

Inst_G_RAM: G_RAM PORT MAP(
                clk => clk,
                rst => rst,
                we => we,
                enaw => enaw,
                enar => enar,
                sw => sw,
                dip => datram,
                addrram => addrram,
                dop => dop
        );

end Behavioral;
```

Lampiran 13 VHDL Graphic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity graphic is
   Port ( clk : in  STD_LOGIC;
rst : in  STD_LOGIC;
pwm : in  STD_LOGIC;
rot : in  STD_LOGIC;
x : in  STD_LOGIC_VECTOR (9 downto 0);
y : in  STD_LOGIC_VECTOR (9 downto 0);
dat : out  STD_LOGIC_VECTOR (0 downto 0);
enaw : out  STD_LOGIC);
end graphic;

architecture Behavioral of graphic is
signal enawep, enawen : STD_LOGIC;
signal pxop, pxon : STD_LOGIC_VECTOR (0
downto 0) := (others => '0');

begin
process(clk, rst) is
begin
if rst = '1' then
        enawep <= '0';
        pxop <= (others => '0');
elsif clk'event and clk = '0' then
        enawep <= enawen;
        pxop <= pxon;
end if;
end process;

process(x, y, pwm, rot, enawep)
begin
enawen <= enawep;
if x <= 479 and y <= 63 then
    if y = 15 then
        if pwm = '1' then
                pxon <= "1";
                enawen <= '1';
        else
                pxon <= "0";
                enawen <= '1';
        end if;
    elsif y = 47 then
        if pwm = '1' then
                pxon <= "0";
                enawen <= '1';
        else
                pxon <= "1";
                enawen <= '1';
        end if;
    else
        pxon <= "0";
        enawen <= '1';
    end if;
elsif x <= 479 and y >= 64 and y <= 127 then
```

```
    if y = 79 then
        if rot = '1' then
                pxon <= "1";
                enawen <= '1';
        else
                pxon <= "0";
                enawen <= '1';
        end if;
    elsif y = 111 then
        if rot = '1' then
                pxon <= "0";
                enawen <= '1';
        else
                pxon <= "1";
                enawen <= '1';
        end if;
    else
        pxon <= "0";
        enawen <= '1';
    end if;
else
    enawen <= '0';
    pxon <= "0";
end if;
end process;

---OUTPUT---
dat <= pxop;
enaw <= enawep;
------------
end Behavioral;
```

Lampiran 14 VHDL MUX

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX is
   Port ( s : in  STD_LOGIC_VECTOR (2 downto
0);
fontpix : in  STD_LOGIC_VECTOR (0 downto 0);
dat : in  STD_LOGIC_VECTOR (0 downto 0);
enawpr : in  STD_LOGIC;
enawt : in  STD_LOGIC;
enawk : in  STD_LOGIC;
enaw : out  STD_LOGIC;
dip : out  STD_LOGIC_VECTOR (0 downto 0));
end MUX;

architecture Behavioral of MUX is
begin
process(s, fontpix, enawt, enawk, dat, enawpr)
begin
if s = 0 then
    dip <= fontpix;
    enaw <= enawt;
elsif s = 1 then
    dip <= fontpix;
    enaw <= enawk;
elsif s = 2 then
    dip <= dat;
    enaw <= enawpr; --pwm
elsif s = 3 then
    dip <= dat;
    enaw <= enawpr; --Rotary
elsif s = 4 then
    dip <= "0";
    enaw <= '1';
elsif s = 5 then
    dip <= "0";
    enaw <= '1';
else
    dip <= "0";
    enaw <= '0';
end if;
end process;
end Behavioral;
```

Lampiran 15 VHDL GRAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;

entity G_RAM is
    Port ( clk : in  STD_LOGIC;
rst : in  STD_LOGIC;
we  : in  STD_LOGIC_VECTOR (0 downto 0);
enaw  : in  STD_LOGIC;
enar  : in  STD_LOGIC;
sw  : in  STD_LOGIC_VECTOR (0 downto 0);
dip : in  STD_LOGIC_VECTOR (0 downto 0);
addrram : in  STD_LOGIC_VECTOR (16 downto
0);
dop : out  STD_LOGIC_VECTOR (0 downto 0));
end G_RAM;

architecture Behavioral of G_RAM is
signal addr : STD_LOGIC_VECTOR(16
DOWNTO 0);
signal ena : STD_LOGIC;
--Deklarasi variabel—

COMPONENT BRAM
  PORT (
clka : IN STD_LOGIC;
rsta : IN STD_LOGIC;
ena : IN STD_LOGIC;
wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
addra : IN STD_LOGIC_VECTOR(16 DOWNTO
0);
dina : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
douta : OUT STD_LOGIC_VECTOR(0 DOWNTO
0));
END COMPONENT;

begin
--wr = '0' membaca --
addr <= addrram
ena <= enaw when sw = 1 else enar;

Inst_BRAM : BRAM
  PORT MAP (
    clka => clk,
    rsta => rst,
    ena => ena,
    wea => we,
    addra => addr,
    dina => dip,
    douta => dop
  );
end Behavioral;
```

Lampiran 16 VHDL VGA *Driver*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_unsigned.all;

entity Inst_VGA_Driver is
    Port ( clk : in  STD_LOGIC;
rst : in  STD_LOGIC;
ri : in  STD_LOGIC_VECTOR (0 downto 0);
ro : out  STD_LOGIC_VECTOR (2 downto 0);
go : out  STD_LOGIC_VECTOR (2 downto 0);
bo : out  STD_LOGIC_VECTOR (2 downto 1);
hs : out  STD_LOGIC;
vs : out  STD_LOGIC;
x : out  STD_LOGIC_VECTOR (9 downto 0);
y : out  STD_LOGIC_VECTOR (9 downto 0));
end Inst_VGA_Driver;

architecture Behavioral of Inst_VGA_Driver is
-----------------Counter HS-----------------------------
signal cntHS_P, cntHS_N : unsigned (9 downto 0)
:= (others => '0'); --Scan Pixel
--------------------------------------------------------
-----------------Counter VS-----------------------------
signal cntVS_P, cntVS_N : unsigned (9 downto 0)
:= (others => '0'); --Scan Line
--------------------------------------------------------
-------------Buffer Output Signal HS------------------
-
signal HS_Buf_P, HS_Buf_N : STD_LOGIC := '0';
--Signal Buffer Synch Pulsa HS (Retrace)
--------------------------------------------------------
-------------Buffer Output Signal VS------------------
-
signal VS_Buf_P, VS_Buf_N : STD_LOGIC := '0';
--Signal Buffer Synch Pulsa VS (Retrace)
--------------------------------------------------------
-----------------Signal Video----------------------------
signal ve : STD_LOGIC; --Scan video
--------------------------------------------------------
----------------RED  OUTPUT-------------------------
---
signal ro_P, ro_N : STD_LOGIC_VECTOR (2
downto 0) := (others => '0'); --RED
--------------------------------------------------------
---------------GREEN  OUTPUT-----------------------
-----
signal go_P, go_N : STD_LOGIC_VECTOR (2
downto 0) := (others => '0'); --GREEN
--------------------------------------------------------
---------------BLUE OUTPUT-------------------------
---
signal bo_P, bo_N : STD_LOGIC_VECTOR (2
downto 1) := (others => '0'); --BLUE
--------------------------------------------------------
begin
----------------------MEMORY-------------------------
-
```

```
process(clk, rst)
begin
if rst = '1' then
        cntHS_P <= (others => '0');
        cntVS_P <= (others => '0');
        HS_Buf_P <= '0';
        VS_Buf_P <= '0';
        ro_P <= (others => '0');
        go_P <= (others => '0');
        bo_P <= (others => '0');
elsif clk'event and clk = '0' then
        cntHS_P <= cntHS_N;
        cntVS_P <= cntVS_N;
        HS_Buf_P <= HS_Buf_N;
        VS_Buf_P <= VS_Buf_N;
        ro_P <= ro_N;
        go_P <= go_N;
        bo_P <= bo_N;
end if;
end process;
----------Video Enable----------
ve <= '1' when cntHS_P >= 0 and cntHS_P <= 639
and cntVS_P >= 0 and cntVS_P <= 479 else '0';
--------------------------------------------------
----------Counter HS and VS----------
process(cntHS_P, cntVS_P)
begin
        cntHS_N <= cntHS_P + 1;
        cntVS_N <= cntVS_P;
----------Kombi Counter
if cntHS_P = 799 then
        cntVS_N <= cntVS_P + 1;
        cntHS_N <= (others => '0');
else
        cntHS_N <= cntHS_P + 1;
end if;
----------Kombi Counter
if cntVS_P = 524 and cntHS_P = 799 then
        cntVS_N <= (others => '0');
end if;
end process;
--------------------------------------------------
----------RGB video----------
process(ve, ri)
begin
if ve = '1' then
        ro_N <= (ri & ri & ri);
        go_N <= "000";
        bo_N <= (ri & ri);
else
        ro_N <= "000";
        go_N <= "000";
        bo_N <= "00";
end if;
end process;
--------------------------------------------------
x <= STD_LOGIC_VECTOR (cntHS_P);
y <= STD_LOGIC_VECTOR (cntVS_P);
----------Buffer Horizontal----------
HS_Buf_N <= '0' when cntHS_P >= 655 and
cntHS_P <= 751 else '1';
--------------------------------------------------
----------Buffer Vertical----------
        VS_Buf_N <= '0' when cntVS_P
>= 489 and cntVS_P <= 491 else '1';
--------------------------------------------------
----------Output Horizontal----------
                hs <= HS_Buf_P;
--------------------------------------------------
----------Output Vertical----------
                vs <= VS_Buf_P;
--------------------------------------------------
----------Output RED----------
                ro <= (ro_P);
--------------------------------------------------
----------Output GREEN----------
-
                go <= (go_P);
--------------------------------------------------
----------Output BLUE----------
                bo <= (bo_P);
--------------------------------------------------
end Behavioral;
```

Lampiran 17 *Testbench Keyboard Driver*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Keyboard_driver_Testbench IS
END Keyboard_driver_Testbench;

ARCHITECTURE behavior OF
Keyboard_driver_Testbench IS

  -- Component Declaration for the Unit Under Test
(UUT)

   COMPONENT Keyboard_driver
   PORT(
clk : IN  std_logic;
rst : IN  std_logic;
ps2_clk : IN  std_logic;
ps2_dat : IN  std_logic;
busy : OUT  std_logic;
port_code : OUT  std_logic_vector(7 downto 0);
code : OUT  std_logic_vector(7 downto 0)
      );
   END COMPONENT;

  --Inputs
  signal clk : std_logic := '0';
  signal rst : std_logic := '0';
  signal ps2_clk : std_logic := '0';
  signal ps2_dat : std_logic := '0';

          --Outputs
  signal busy : std_logic;
  signal port_code : std_logic_vector(7 downto 0);
  signal code : std_logic_vector(7 downto 0);

  -- Clock period definitions
  constant clk_period : time := 40 ns;
  constant ps2_clk_period : time := 100 us;

BEGIN

          -- Instantiate the Unit Under Test (UUT)
  uut: Keyboard_driver PORT MAP (
        clk => clk,
        rst => rst,
        ps2_clk => ps2_clk,
        ps2_dat => ps2_dat,
        busy => busy,
        port_code => port_code,
        code => code
      );

  -- Clock process definitions
  clk_process :process
  begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
end process;
ps2_clk_process :process
begin
                ps2_clk <= '0';
                wait for ps2_clk_period/2;
                ps2_clk <= '1';
                wait for ps2_clk_period/2;
end process;

-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
  wait for 40 ns;
                rst <= '0';
                ps2_dat <= '0';
                wait for ps2_clk_period;
                ps2_dat <= '0';
                wait for ps2_clk_period;
                ps2_dat <= '1';
                wait for ps2_clk_period;
                ps2_dat <= '1';
                wait for ps2_clk_period;
                ps2_dat <= '0';
                wait for ps2_clk_period;
                ps2_dat <= '1';
                wait for ps2_clk_period;
                ps2_dat <= '0';
                wait for ps2_clk_period;
                ps2_dat <= '0';
                wait for ps2_clk_period;
                ps2_dat <= '0';
                wait for ps2_clk_period;
                ps2_dat <= '0';
                wait for ps2_clk_period;
                ps2_dat <= '1';
                wait for ps2_clk_period;
                ps2_dat <= '1';
  -- insert stimulus here

  wait;
end process;
END;
```

Lampiran 18 *Testbench Keyboard Converter*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Keyboard_converter_Testbench IS
END Keyboard_converter_Testbench;

ARCHITECTURE behavior OF
Keyboard_converter_Testbench IS

-- Component Declaration for the Unit Under Test
(UUT)

  COMPONENT Keyboard_converter
  PORT(
     code : IN  std_logic_vector(7 downto 0);
     ascii : OUT  std_logic_vector(7 downto 0)
    );
  END COMPONENT;

  --Inputs
  signal code : std_logic_vector(7 downto 0) :=
(others => '0');

        --Outputs
  signal ascii : std_logic_vector(7 downto 0);
  -- No clocks detected in port list. Replace
<clock> below with
  -- appropriate port name

  constant clock_period : time := 100 ns;

BEGIN
        -- Instantiate the Unit Under Test (UUT)
  uut: Keyboard_converter PORT MAP (
      code => code,
      ascii => ascii
    );

 -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 100 ns;
                code <= x"f0";
    wait for 100 ns;
                code <= x"16";
                wait for 100 ns;
                code <= x"f0";
                wait for 100 ns;
                code <= x"45";
    -- insert stimulus here
    wait;
  end process;

END;
```

Lampiran 19 *Testbench Scan Code*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

USE ieee.numeric_std.ALL;
 ENTITY scan_code_Testbench IS
END scan_code_Testbench;

ARCHITECTURE behavior OF
scan_code_Testbench IS
-- Component Declaration for the Unit Under Test
(UUT)
    COMPONENT Inst_scan_code
   PORT(
       clk : IN  std_logic;
       rst : IN  std_logic;
       ascii : IN  std_logic_vector(7 downto 0);
       busy : IN  std_logic;
       x : IN  std_logic_vector(9 downto 0);
       y : IN  std_logic_vector(9 downto 0);
       enar : OUT  std_logic;
       we : OUT  std_logic_vector(0 downto 0);
       sw : OUT  std_logic_vector(0 downto 0);
       enawk : OUT  std_logic;
       enawt : OUT  std_logic;
       s : OUT  std_logic_vector(2 downto 0);
       addrram : OUT  std_logic_vector(16 downto
0);
       font_pix : OUT  std_logic_vector(0 downto 0)
      );
    END COMPONENT;

   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';
   signal ascii : std_logic_vector(7 downto 0) :=
(others => '0');
   signal busy : std_logic := '0';
   signal x : std_logic_vector(9 downto 0) := (others
=> '0');
   signal y : std_logic_vector(9 downto 0) := (others
=> '0');

          --Outputs
   signal enar : std_logic;
   signal we : std_logic_vector(0 downto 0);
   signal sw : std_logic_vector(0 downto 0);
   signal enawk : std_logic;
   signal enawt : std_logic;
   signal s : std_logic_vector(2 downto 0);
   signal addrram : std_logic_vector(16 downto 0);
   signal font_pix : std_logic_vector(0 downto 0);

   -- Clock period definitions
   constant clk_period : time := 40 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
```

```
uut: Inst_scan_code PORT MAP (
      clk => clk,
      rst => rst,
      ascii => ascii,
      busy => busy,
      x => x,
      y => y,
      enar => enar,
      we => we,
      sw => sw,
      enawk => enawk,
      enawt => enawt,
      s => s,
      addrram => addrram,
      font_pix => font_pix
    );

-- Clock process definitions
clk_process :process
begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
end process;

-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
  wait for 40 ns;
                rst <= '0';
  wait for clk_period*5;
                busy <= '1';
  -- insert stimulus here
                wait for clk_period*2;
                ascii <= x"31";
                wait for clk_period*5;
                busy <= '0';
                wait for clk_period*5;
                busy <= '1';
                wait for clk_period*2;
                ascii <= x"31";
                wait for clk_period*5;
                busy <= '0';
                y <= "0000000011";
                x <= "1000010000";
                wait for clk_period;
                        x <= "1000010001";
                wait for clk_period;
                        x <= "1000010010";
                wait for clk_period;
                        x <= "1000010011";
                wait for clk_period;
                        x <= "1000010100";
                wait for clk_period;
                        x <= "1000010101";
                wait for clk_period;
                        x <= "1000010110";
                wait for clk_period;
                        x <= "1000010111";
                        -----------------------
                wait for clk_period;
                        x <= "1000000000";
                wait for clk_period;
                        x <= "1000000001";
                wait for clk_period;
                        x <= "1000000010";
                wait for clk_period;
                        x <= "1000000011";
                wait for clk_period;
                        x <= "1000000100";
                wait for clk_period;
                        x <= "1000000101";
                wait for clk_period;
                        x <= "1000000110";
                wait for clk_period;
                        x <= "1000000111";
                wait for clk_period*8;
--
    wait;
  end process;

END;
```

Lampiran 20 *Testbench Scan* PWM dan *Rotary*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Scan_PWM_Rotary_Testbench IS
END Scan_PWM_Rotary_Testbench;

ARCHITECTURE behavior OF Scan_PWM_Rotary_Testbench IS

  -- Component Declaration for the Unit Under Test (UUT)

  COMPONENT Scan_PWM_Rotary
  PORT(
       clk : IN  std_logic;
       rst : IN  std_logic;
       sinyal_pwm : IN  std_logic;
       sinyal_Rotary : IN  std_logic;
       puls_pwm : OUT  std_logic;
       puls_Rotary : OUT  std_logic
      );
  END COMPONENT;

 --Inputs
 signal clk : std_logic := '0';
 signal rst : std_logic := '0';
 signal sinyal_pwm : std_logic := '0';
 signal sinyal_Rotary : std_logic := '0';

        --Outputs
 signal puls_pwm : std_logic;
 signal puls_Rotary : std_logic;

 -- Clock period definitions
 constant clk_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
 uut: Scan_PWM_Rotary PORT MAP (
        clk => clk,
        rst => rst,
        sinyal_pwm => sinyal_pwm,
        sinyal_Rotary => sinyal_Rotary,
        puls_pwm => puls_pwm,
        puls_Rotary => puls_Rotary
      );

 -- Clock process definitions
 clk_process :process
 begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
 end process;
  -- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
  wait for 100 ns;
                rst <= '0';
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*1000000;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*1000000;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
                sinyal_pwm <= '0';
```

```
                sinyal_Rotary <= '0';
                wait for clk_period*25;
                sinyal_pwm <= '1';
                sinyal_Rotary <= '1';
                wait for clk_period*25;
    -- insert stimulus here
    wait;
  end process;

END;
```

Lampiran 21 *Testbench* VRAM

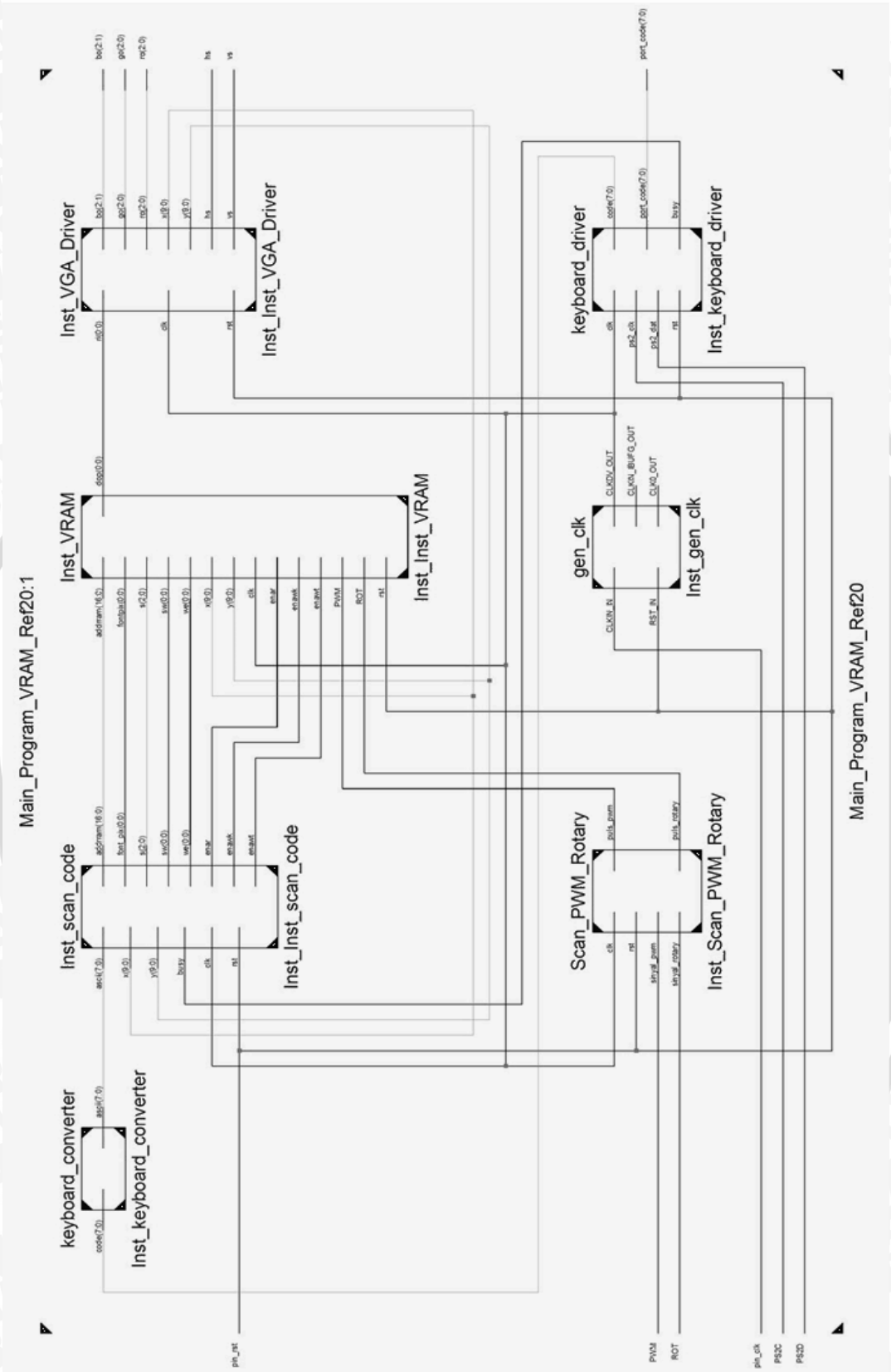
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

USE ieee.numeric_std.ALL;

ENTITY VRAM_Testbench IS
END VRAM_Testbench;

ARCHITECTURE behavior OF VRAM_Testbench
IS

   -- Component Declaration for the Unit Under
Test (UUT)

   COMPONENT Inst_VRAM
   PORT(
        clk : IN  std_logic;
        rst : IN  std_logic;
        PWM : IN  std_logic;
        ROT : IN  std_logic;
        fontpix : IN  std_logic_vector(0 downto 0);
        enawk : IN  std_logic;
        enawt : IN  std_logic;
        enar : IN  std_logic;
        we : IN  std_logic_vector(0 downto 0);
        sw : IN  std_logic_vector(0 downto 0);
        s : IN  std_logic_vector(2 downto 0);
        addrram : IN  std_logic_vector(16 downto 0);
        x : IN  std_logic_vector(9 downto 0);
        y : IN  std_logic_vector(9 downto 0);
        dop : OUT  std_logic_vector(0 downto 0)
       );
   END COMPONENT;

   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';
   signal PWM : std_logic := '0';
   signal ROT : std_logic := '0';
   signal fontpix : std_logic_vector(0 downto 0) :=
(others => '0');
   signal enawk : std_logic := '0';
   signal enawt : std_logic := '0';
   signal enar : std_logic := '0';
   signal we : std_logic_vector(0 downto 0) :=
(others => '0');
   signal sw : std_logic_vector(0 downto 0) :=
(others => '0');
   signal s : std_logic_vector(2 downto 0) := (others
=> '0');
   signal addrram : std_logic_vector(16 downto 0)
:= (others => '0');
   signal x : std_logic_vector(9 downto 0) := (others
=> '0');
   signal y : std_logic_vector(9 downto 0) := (others
=> '0');

        --Outputs
   signal dop : std_logic_vector(0 downto 0);
```

```
  -- Clock period definitions
  constant clk_period : time := 40 ns;

BEGIN
 -- Instantiate the Unit Under Test (UUT)
  uut: Inst_VRAM PORT MAP (
        clk => clk,
        rst => rst,
        PWM => PWM,
        ROT => ROT,
        fontpix => fontpix,
        enawk => enawk,
        enawt => enawt,
        enar => enar,
        we => we,
        sw => sw,
        s => s,
        addrram => addrram,
        x => x,
        y => y,
        dop => dop
      );

  -- Clock process definitions
  clk_process :process
  begin
          clk <= '0';
          wait for clk_period/2;
          clk <= '1';
          wait for clk_period/2;
  end process;

  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 40 ns;
          rst <= '0';
    wait for clk_period*2;
          we <= "1";
          sw <= "1";
          enar <= '0';
          s <= "010";
    -- insert stimulus here
    wait for clk_period;
          addrram <= "00011110000000000";
          y <= "0000001111";
          x <= "0000000000";
          pwm <= '1';
    wait for clk_period;
          addrram <= "00011110000000001";
          y <= "0000001111";
          x <= "0000000001";
          pwm <= '1';
    wait for clk_period;
          addrram <= "00011110000000010";
          y <= "0000001111";
          x <= "0000000010";
          pwm <= '1';
    wait for clk_period;
          addrram <= "00011110000000011";
          y <= "0000001111";
          x <= "0000000011";
          pwm <= '1';
    wait for clk_period;
          addrram <= "00011110000000100";
          y <= "0000001111";
          x <= "0000000100";
          pwm <= '0';
    wait for clk_period;
          addrram <= "00011110000000101";
          y <= "0000001111";
          x <= "0000000101";
          pwm <= '0';
    wait for clk_period;
          addrram <= "00011110000000110";
          y <= "0000001111";
          x <= "0000000110";
          pwm <= '0';
    wait for clk_period;
          addrram <= "00011110000000111";
          y <= "0000001111";
          x <= "0000000111";
          pwm <= '0';
    wait for clk_period;
          addrram <= "00011110000000000";
          we <= "0";
          sw <= "0";
          enar <= '1';
          pwm <= '1';
    wait for clk_period;
          addrram <= "00011110000000001";
    wait for clk_period;
          addrram <= "00011110000000010";
    wait for clk_period;
          addrram <= "00011110000000011";
    wait for clk_period;
          addrram <= "00011110000000100";
    wait for clk_period;
          addrram <= "00011110000000101";
    wait for clk_period;
          addrram <= "00011110000000110";
    wait for clk_period;
          addrram <= "00011110000000111";
     wait;
   end process;

END;
```

Lampiran 22 *Testbench* VGA *Driver*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

USE ieee.numeric_std.ALL;

ENTITY VGA_Driver_Testbench IS
END VGA_Driver_Testbench;

ARCHITECTURE behavior OF
VGA_Driver_Testbench IS

-- Component Declaration for the Unit Under Test
(UUT)

  COMPONENT Inst_VGA_Driver
  PORT(
     clk : IN  std_logic;
     rst : IN  std_logic;
     ri : IN  std_logic_vector(0 downto 0);
     ro : OUT  std_logic_vector(2 downto 0);
     go : OUT  std_logic_vector(2 downto 0);
     bo : OUT  std_logic_vector(2 downto 1);
     hs : OUT  std_logic;
     vs : OUT  std_logic;
     x : OUT  std_logic_vector(9 downto 0);
     y : OUT  std_logic_vector(9 downto 0)
    );
  END COMPONENT;

  --Inputs
  signal clk : std_logic := '0';
  signal rst : std_logic := '0';
  signal ri : std_logic_vector(0 downto 0) := (others
=> '0');

        --Outputs
  signal ro : std_logic_vector(2 downto 0);
  signal go : std_logic_vector(2 downto 0);
  signal bo : std_logic_vector(2 downto 1);
  signal hs : std_logic;
  signal vs : std_logic;
  signal x : std_logic_vector(9 downto 0);
  signal y : std_logic_vector(9 downto 0);

  -- Clock period definitions
  constant clk_period : time := 40 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
  uut: Inst_VGA_Driver PORT MAP (
       clk => clk,
       rst => rst,
       ri => ri,
       ro => ro,
       go => go,
       bo => bo,
       hs => hs,
       vs => vs,
       x => x,
       y => y
      );

-- Clock process definitions
clk_process :process
begin
       clk <= '0';
       wait for clk_period/2;
       clk <= '1';
       wait for clk_period/2;
end process;

-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
  wait for 40 ns;
                rst <= '0';
  wait for clk_period*10;
                ri <= "1";
  -- insert stimulus here

  wait;
end process;

END;
```

Lampiran 23 Skematik VHD Sistem Pemonitoran



USB Port

The Nexys2 includes a high-speed USB2 port based on a Cypress CY7C68013A USB controller. The USB port can be used to program the on-board Xilinx devices, to perform user-data transfers at up to 38Mbytes/sec, and to provide power to the board. Programming is accomplished with Digilent's free Adept Suite Software. User data transfers can also be accomplished using the Adept software, or custom user software can be written using Digilent's public API's to access the Nexys2 USB connection. Information on using Adept and/or the public API's to transfer data can be found on the Digilent website.

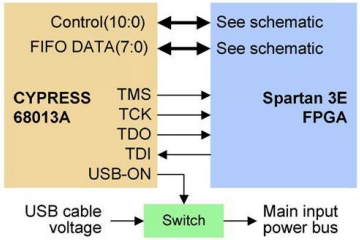


Figure 11: Nexys2 USB circuit

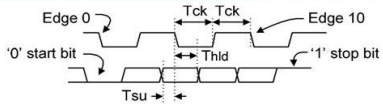
The USB port can also provide power to the Nexys2 board if the power select jumper is set to "USB". The USB specification requires that attached devices draw no more than 100mA until they have requested more current, after which up to 500mA may be drawn. When first attached to a USB host, the Nexys2 board requests 500mA, and then activates a transistor switch to connect the USB cable voltage to the main input power bus. The Nexys2 board typically draws around 300mA from the USB cable, and care should be taken (especially when using peripheral boards) to ensure that no more than 500mA is drawn.

PS/2 Port

The 6-pin mini-DIN connector can accommodate a PS/2 mouse or keyboard. Most PS/2 devices can operate from a 3.3V supply, but older devices may require a 5VDC supply. A three-pin jumper on the Nexys2 board immediately adjacent to the PS/2 connector selects whether regulated 3.3V or the main input power bus voltage (VU) is supplied to the PS/2 connector. To send 5V to the PS/2 connector, set the PS2 power jumper to Vswt (the main input power bus), and ensure the board is powered from USB or a 5VDC wall-plug supply. To send 3.3V to the connector, set the jumper to 3.3V.

Figure 12: Nexys2 PS/2 circuits

Both the mouse and keyboard use a two-wire serial bus (clock and data) to communicate with a host device. Both use 11-bit words that include a start, stop and odd parity bit, but the data packets are organized differently, and the keyboard interface allows bi-directional data transfers (so the host device can illuminate state LEDs on the keyboard). Bus timings are shown in the figure. The clock and data signals are only driven when data transfers occur, and otherwise they are held in the "idle" state at logic '1'. The timings define signal requirements for mouse-to-host communications and bi-directional keyboard communications. A PS/2 interface circuit can be implemented in the FPGA to create a keyboard or mouse interface.



Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us

Figure 13: PS/2 signal timings

Keyboard

The keyboard uses open-collector drivers so the keyboard or an attached host device can drive the two-wire bus (if the host device will not send data to the keyboard, then the host can use input-only ports).

PS2-style keyboards use scan codes to communicate key press data. Each key is assigned a code that is sent whenever the key is pressed; if the key is held down, the scan code will be sent repeatedly about once every 100ms. When a key is released, a "F0" key-up code is sent, followed by the scan code of the released key. If a key can be "shifted" to produce a new character (like a capital letter), then a shift character is sent in addition to the scan code, and the host must determine which ASCII character to use. Some keys, called extended keys, send an "E0" ahead of the scan code (and they may send more than one scan code). When an extended key is released, an "E0 F0" key-up code is sent, followed by the scan code. Scan codes for most keys are shown in the figure. A host device can also send data to the keyboard. Below is a short list of some common commands a host might send.

ED Set Num Lock, Caps Lock, and Scroll Lock LEDs. Keyboard returns "FA" after receiving "ED", then host sends a byte to set LED status: Bit 0 sets Scroll Lock; bit 1 sets Num Lock; and Bit 2 sets Caps lock. Bits 3 to 7 are ignored.
EE Echo (test). Keyboard returns "EE" after receiving "EE".
F3 Set scan code repeat rate. Keyboard returns "F3" on receiving "FA", then host sends second byte to set the repeat rate.
FE Resend. "FE" directs keyboard to re-send most recent scan code.
FF Reset. Resets the keyboard.

The keyboard can send data to the host only when both the data and clock lines are high (or idle). Since the host is the "bus master", the keyboard must check to see whether the host is sending data before driving the bus. To facilitate this, the clock line is used as a "clear to send" signal. If the host pulls the clock line low, the keyboard must not send any data until the clock is released. The keyboard sends data to the host in 11-bit words that contain a '0' start bit, followed by 8-bits of scan code (LSB first), followed by an odd parity bit and terminated with a '1' stop bit. The keyboard generates 11 clock transitions (at around 20 - 30KHz) when the data is sent, and data is valid on the falling edge of the clock.

Scan codes for most PS/2 keys are shown in the figure below.

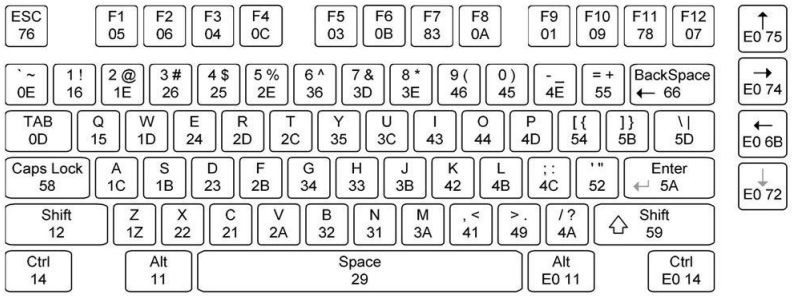


Figure 14: PS/2 keyboard scan codes

Mouse

The mouse outputs a clock and data signal when it is moved; otherwise, these signals remain at logic '1'. Each time the mouse is moved, three 11-bit words are sent from the mouse to the host device. Each of the 11-bit words contains a '0' start bit, followed by 8 bits of data (LSB first), followed by an odd parity bit, and terminated with a '1' stop bit. Thus, each data transmission contains 33 bits, where bits 0, 11, and 22 are '0' start bits, and bits 10, 21, and 33 are '1' stop bits. The three 8-bit data fields contain movement data as shown in the figure above. Data is valid at the falling edge of the clock, and the clock period is 20 to 30KHz.

The mouse assumes a relative coordinate system wherein moving the mouse to the right generates a positive number in the X field, and moving to the left generates a negative number. Likewise, moving the mouse up generates a positive number in the Y field, and moving down represents a negative number (the XS and YS bits in the status byte are the sign bits – a '1' indicates a negative number). The magnitude of the X and Y numbers represent the rate of mouse movement – the larger the number, the faster the mouse is moving (the XV and YV bits in the status byte are movement overflow indicators – a '1' means overflow has occurred). If the mouse moves continuously, the 33-bit transmissions are repeated every 50ms or so. The L and R fields in the status byte indicate Left and Right button presses (a '1' indicates the button is being pressed).

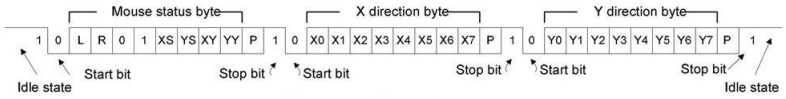
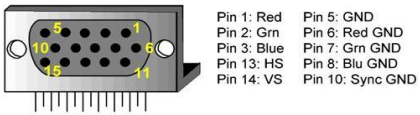


Figure 15: Mouse data format

VGA Port

The Nexys2 board uses 10 FPGA signals to create a VGA port with 8-bit color and the two standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync). The color signals use resistor-divider circuits that work in conjunction with the 75-ohm termination resistance of the VGA display to create eight signal levels on the red and green VGA signals, and four on blue (the human eye is less sensitive to blue levels). This circuit, shown in figure 13, produces video color signals that proceed in equal increments between 0V (fully off) and 0.7V (fully on). Using this circuit, 256 different colors can be displayed, one for each unique 8-bit pattern. A video controller circuit must be created in the FPGA to drive the sync and color signals with the correct timing in order to produce a working display system.



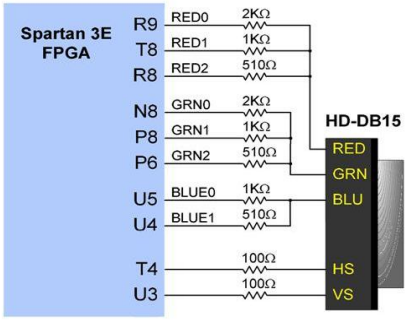


Figure 16: VGA pin definitions and Nexys2 circuit

VGA System Timing

VGA signal timings are specified, published, copyrighted and sold by the VESA organization (www.vesa.org). The following VGA system timing information is provided as an example of how a VGA monitor might be driven in 640 by 480 mode. For more precise information, or for information on other VGA frequencies, refer to documentation available at the VESA website.

CRT-based VGA displays use amplitude-modulated moving electron beams (or cathode rays) to display information on a phosphor-coated screen. LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel-by-pixel basis. Although the following description is limited to CRT displays, LCD displays have evolved to use the same signal timings as CRT displays (so the "signals" discussion below pertains to both CRTs and LCDs). Color CRT displays use three electron beams (one for red, one for blue, and one for green) to energize the phosphor that coats the inner side of the display end of a cathode ray tube (see illustration). Electron beams emanate from "electron guns" which are finely-pointed heated cathodes placed in close proximity to a positively charged annular plate called a "grid". The electrostatic force imposed by the grid pulls rays of energized electrons

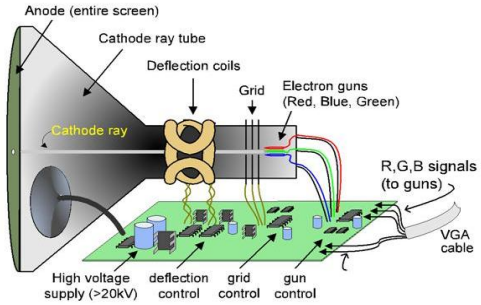


Figure 17: CRT deflection system

from the cathodes, and those rays are fed by the current that flows into the cathodes. These particle rays are initially accelerated towards the grid, but they soon fall under the influence of the much larger electrostatic force that results from the entire phosphor-coated display surface of the CRT being charged to 20kV (or more). The rays are focused to a fine beam as they pass through the center of the grids, and then they accelerate to impact on the phosphor-coated display surface. The phosphor surface glows brightly at the impact point, and it continues to glow for several hundred microseconds after the beam is removed. The larger the current fed into the cathode, the brighter the phosphor will glow.

Between the grid and the display surface, the beam passes through the neck of the CRT where two coils of wire produce orthogonal electromagnetic fields. Because cathode rays are composed of charged particles (electrons), they can be deflected by these magnetic fields. Current waveforms are passed through the coils to produce magnetic fields that interact with the cathode rays and cause them to transverse the display surface in a "raster" pattern, horizontally from left to right and vertically from top to bottom. As the cathode ray moves over the surface of the display, the current sent to the electron guns can be increased or decreased to change the brightness of the display at the cathode ray impact point.

Information is only displayed when the beam is moving in the "forward" direction (left to right and top to bottom), and not during the time the beam is reset back to the left or top edge of the display. Much of the potential display time is therefore lost in "blanking" periods when the beam is reset and stabilized to begin a new horizontal or vertical display pass. The size of the beams, the frequency at which the beam can be traced across the display, and the frequency at which the electron beam can be modulated determine the display resolution. Modern VGA displays can accommodate different resolutions, and a VGA controller circuit dictates the resolution by producing timing signals to control the raster patterns. The controller must produce synchronizing pulses at 3.3V (or 5V) to set the frequency at which current flows through the deflection coils, and it must ensure that video data is applied to the electron guns at the correct time. Raster video displays define a number of "rows" that corresponds to the number of horizontal passes the cathode makes over the display area, and a number of "columns" that corresponds to an area on each row that is assigned to one "picture element" or pixel. Typical displays use from 240 to 1200 rows and from 320 to 1600 columns. The overall size of a display and the number of rows and columns determines the size of each pixel.

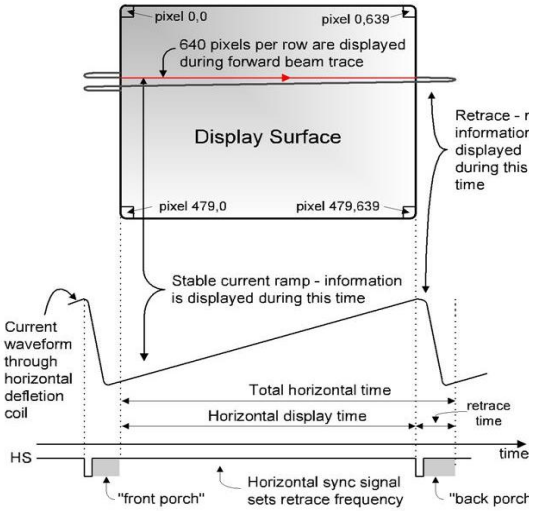
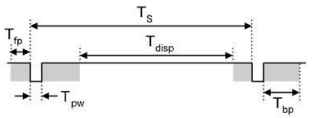


Figure 18: VGA system signals

Video data typically comes from a video refresh memory, with one or more bytes assigned to each pixel location (the Nexys2 uses three bits per pixel). The controller must index into video memory as the beams move across the display, and retrieve and apply video data to the display at precisely the time the electron beam is moving across a given pixel.

A VGA controller circuit must generate the HS and VS timings signals and coordinate the delivery of video data based on the pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the "refresh" frequency of the display, or the frequency at which all information on the display is redrawn. The minimum refresh frequency is a function of the display's phosphor and electron beam intensity, with practical refresh frequencies falling in the 50Hz to 120Hz range. The number of lines to be displayed at a given refresh frequency defines the horizontal "retrace" frequency. For a 640-pixel by 480-row display using a 25MHz pixel clock and 60 +/-1Hz refresh, the signal timings shown in the table at right can be derived. Timings for sync pulse width and front and back porch intervals (porch intervals are the pre- and post-sync pulse times during which information cannot be displayed) are based on observations taken from actual VGA displays.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 19: VGA system timings for 640x480 display

A VGA controller circuit decodes the output of a horizontal-sync counter driven by the pixel clock to generate HS signal timings. This counter can be used to locate any pixel location on a given row. Likewise, the output of a vertical-sync counter that increments with each HS pulse can be used to generate VS signal timings, and this counter can be used to locate any given row. These two continually running counters can be used to form an address into video RAM. No time relationship between the onset of the HS pulse and the onset of the VS pulse is specified, so the designer can arrange the counters to easily form video RAM addresses, or to minimize decoding logic for sync pulse generation.

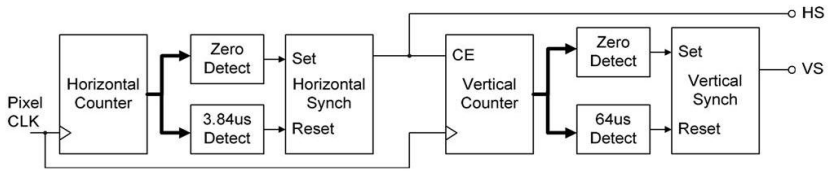


Figure 20: Schematic for a VGA controller circuit

Optional Pipeline Stages

Pipeline stages are currently not supported.

Memory Initialization Capability

The memory contents can be optionally initialized using a memory coefficient (COE) file or by specifying a default data value. A COE file can define the initial contents of each individual memory location, while the default data value option defines the initial content for all locations.

Simulation Models

The Block Memory Generator core provides behavioral and structural simulation models in VHDL and Verilog to give the user the option to perform either simple or precise modeling of memory behaviors, respectively.

Block Memory Generator Functional Description

The Block Memory Generator is used to build custom memory modules from block RAM primitives in Xilinx FPGAs. The core implements an optimal memory by arranging block RAM primitives based on user selections, automating the process of primitive instantiation and concatenation. Using the CORE Generator Graphical User Interface (GUI), users can configure the core and rapidly generate a highly optimized custom memory solution.

Memory Type

The Block Memory Generator creates five memory types: Single-port RAM, Simple Dual-port RAM, True Dual-port RAM, Single-port ROM, and Dual-port ROM. Figure 18 through Figure 22 illustrate the signals available for each type. Optional pins are displayed in italics.

For each configuration, optimizations are made within the core to minimize the total resources used. For example, a Simple Dual-port RAM with symmetric ports can utilize the special Simple Dual-port RAM primitive in Virtex-5 devices, which can save as much as fifty percent of the block RAM resources for memories 512 words deep or fewer. The Single-port ROM allows Read access to the memory space through a single port, as illustrated in Figure 18.

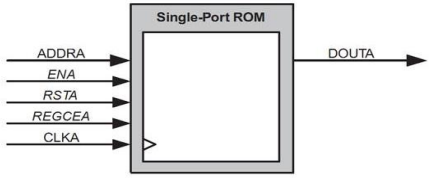


Figure 18: **Single-port ROM**

The Dual-port ROM allows Read access to the memory space through two ports, as shown in Figure 19.

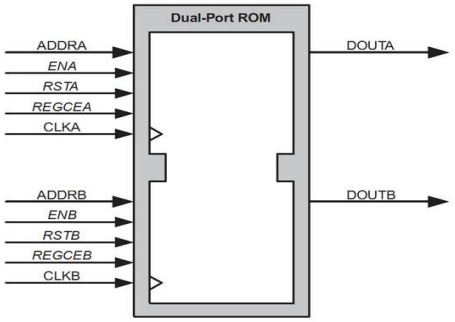


Figure 19: **Dual-port ROM**

The Single-port RAM allows Read and Write access to the memory through a single port, as shown in Figure 20.

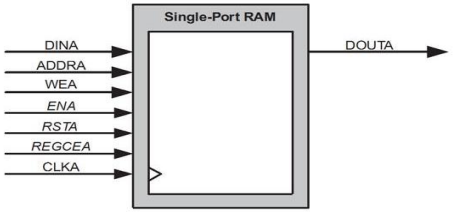


Figure 20: **Single-port RAM**

The Simple Dual-port RAM provides two ports, A and B, as illustrated in Figure 21. Write access to the memory is allowed via port A, and Read access is allowed via port B.

Note: For Virtex family architectures, Read access is via port A and Write access is via port B.

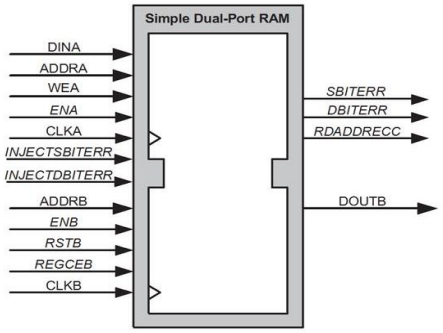


Figure 21: **Simple Dual-port RAM**

The True Dual-port RAM provides two ports, A and B, as illustrated in Figure 22. Read and Write accesses to the memory are allowed on either port.

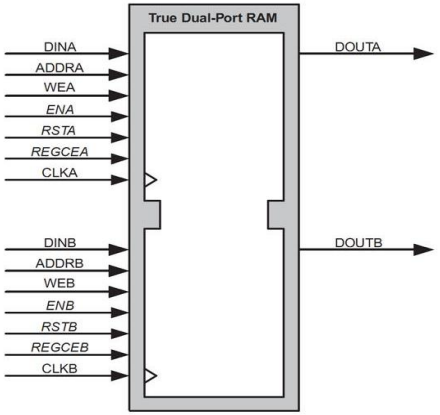


Figure 22: **True Dual-port RAM**

The operating modes have an effect on the relationship between the A and B ports when the A and B port addresses have a collision. For detailed information about collision behavior, see Collision Behavior, page 30. For more information about operating modes, see the block RAM section of the user guide specific to the device family.

- **Write First Mode:** In WRITE_FIRST mode, the input data is simultaneously written into memory and driven on the data output, as shown in Figure 26. This transparent mode offers the flexibility of using the data output bus during a Write operation on the same port.

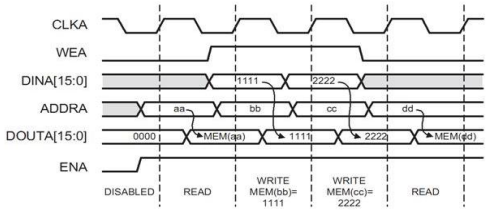


Figure 26: **Write First Mode Example**

Note: The WRITE_FIRST operation is affected by the optional byte-Write feature in Kintex-7, Virtex-7, Virtex-6, Virtex-5, Virtex-4, Spartan-6 and Spartan-3A/3A DSP devices. It is also affected by the optional Read-to-Write aspect ratio feature in Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 devices. For detailed information, see Write First Mode Considerations, page 30.

- **Read First Mode:** In READ_FIRST mode, data previously stored at the Write address appears on the data output, while the input data is being stored in memory. This Read-before-Write behavior is illustrated in Figure 27.

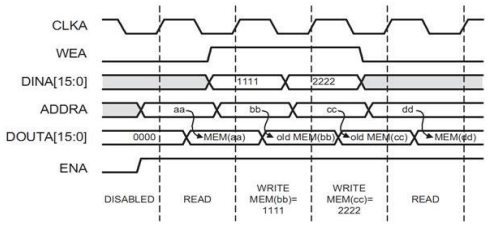


Figure 27: **Read First Mode Example**

- **No Change Mode:** In NO_CHANGE mode, the output latches remain unchanged during a Write operation. As shown in Figure 28, the data output is still the previous Read data and is unaffected by a Write operation on the same port.

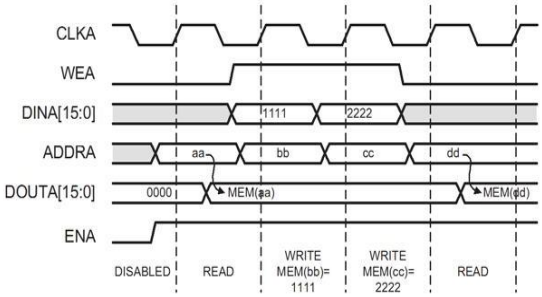


Figure 28: **No Change Mode Example**

Data Width Aspect Ratios

The Block Memory Generator supports data width aspect ratios. This allows the port A data width to be different than the port B data width, as described in Port Aspect Ratios in the following section. In Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA-based memories, all four data buses (DINA, DOUTA, DINB, and DOUTB) can have different widths, as described in Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 Read-to-Write Aspect Ratios, page 28.

The limitations of the data width aspect ratio feature (some of which are imposed by other optional features) are described in Aspect Ratio Limitations, page 29. The CORE Generator GUI ensures only valid aspect ratios are selected.

Port Aspect Ratios

The Block Memory Generator supports port aspect ratios of 1:32, 1:16, 1:8, 1:4, 1:2, 1:1, 2:1, 4:1, 8:1, 16:1, and 32:1. The port A data width can be up to 32 times larger than the port B data width, or vice versa. The smaller data words are arranged in little-endian format, as illustrated in Figure 29.

Port Aspect Ratio Example

Consider a True Dual-port RAM of 32x2048, which is the A port width and depth. From the perspective of an 8-bit B port, the depth would be 8192. The ADDRA bus is 11 bits, while the ADDRB bus is 13 bits. The data is stored little-endian, as shown in Figure 29. Note that A_n is the data word at address *n*, with respect to the A port. B_n is the data word at address n with respect to the B port. A_0 is comprised of B_3, B_2, B_1, and B_0.

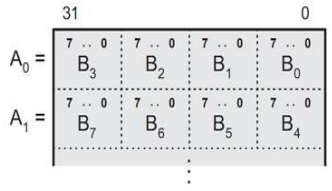


Figure 29: **Port Aspect Ratio Example Memory Map**