PERANCANGAN *CODE GENERATOR* UNTUK MIKROKONTROLER *"BASIC STAMP"* BERBASIS DIAGRAM KEADAAN PADA SISTEM SEKUENSIAL

Eko Pramono[1]. Thomas Sri Widodo[2], Rudi Hartanto[3]
[1]STMIK AMIKOM Yogyakarta
[2]Fakultas Teknik UGM Yogyakarta
[3] Fakultas Teknik UGM Yogyakarta

Abstract

Usage and growth of microcontroller 's world at the last years is very fast. Started from microcontroller using low level language until high level language. For the reason the way of pro gram every microcontroller become very differ.

Microcontroller programming is not easy for a beginner, for this reason, this research will making an application program to facilitate in programmed a microcontroller specially at "Basic Stamp" microcontroller.

This Research will build a Code Generator application, this application programming based on State Diagram, then turned into command line of microcontroller 's instructions.

Although yielded longer command line and use more memory spaces, but microcontroller programming using this Code Generator is faster compared to conventionally programming.

The generator's code application can give new discourse in Basic Stamp microcontroller programming, and very suited for sequential system.

***Kata Kunci**: Microcontroller, Basic Stamp, Code Generator, State Diagram*

1. Pendahuluan

Pada beberapa tahun terakhir ini perbincangan mengenai dunia mikrokontroler semakin marak dan menarik, hal ini dikarenakan kemampuan mikrokontroler itu sendiri yang cukup mengagumkan. Mikrokontroler adalah sebuah komputer kecil yang diciptakan untuk mengatasi permasalahan

dalam perancangan dan pembuatan prototip elektronika dengan cara memperkenalkan sebuah konsep pemuatan program ke dalam mikrokontroler tersebut (**Hernando Barragan, 2004**).

Banyak ragam mikrokontroler yang beredar di pasaran seperti ATMEL. AVR. PlC. LEGO dan lain-lain. Mikrokontroler memiliki keunikan sendiri-sendiri, khususnya dalam hal pemrogramannya. Pemrograman mikrokontroler sampai saat mi kebanyakan masih berbasis baris perintah (*source code*) sebagai suatu perwujudan dan keadaan (state) yang akan dan harus dilaksanakan oleh mikrokontroler tersebut.

Ada banyak pemula yang berbeda disiplin ilmu namun berpotensi dalam mendesain, membuat prototip dan memprogram mikrokontroler, mereka menginginkan suatu cara pemrograman yang mudah serta dapat menguji prototip yang mereka buat sampai pada kemungkinan aspek pengembangannya dan sebuah gagasan yang benar-benar baru (**Hernando Barragan, 2004**).

Dalam penelitian ini akan dibangun suatu aplikasi yang mengimplementasikan keadaan (state) dan langkah-langkah sistem yang menggunakan mikrokontroler untuk diubah ke dalam baris perintah mikrokontroler tersebut, sehingga memudahkan dalam memprogram mikrokontroler tersebut, sebab hanya dengan membangun *State Diagram* maka dapat langsung dibangkitkan baris

perintahnya. Untuk membangun aplikasi mi dibutuhkan bahasa pemrograrnan yang berorientasi obyek. Karena masing keadaan dan langkah-langkah yang harus dilaksanakan oleh mikrokontroler dapat diwujudkan sebagai suatu obyek yang saling berhubungan satu dengan lainnya.

2. Pembahasan

Banyak pengguna dan perancang sistem yang berbasis mikrokontroler mengalami kesulitan ketika harus memprogram suatu mikrokontroler. Walaupun mereka sangat paham mengenai tahapan keadaan yang akan diprogram dalam mikrokontroler tersebut. Hal ini disebabkan kerumitan dan bahasa pemrograman rnikrokontroler itu sendiri yang banyak menggunakan bahasa tingkat rendah atapun bahasa tingkat menengah,

sehingga sering mengakibatkan kerumitan dalam pemrograman mikrokontroler tersebut.

Pembuatan aplikasi code generator (Pembangkit Baris Perintah) ini diupayakan mampu membantu memecahkan kerumitan tersebut. Karena aplikasi mi menggunakan *State Diagram* dalam memprogram mikrokontroler tersebut, sehingga menjadi mudah dalam memprogram mikrokontroler tersebut karena bersifat visual.

Pemrograman mikrokontroler terkadang menjadi sangat rumit karena jenis perintahnya yang cukup banyak dan kurang terstruktur. Oleh karenanya tidak mudah bagi pemula untuk memahaminya dalam waktu singkat.

Suatu sistem yang diterapkan kepada pemula diharapkan dapat memudahkan konsep pembelajaran yang dapat dipetakan dan bermanfaat di dalam konteks yang berbeda. Sehingga dengan demikian, seorang pemula tidak perlu memiliki ketrampi Ian pemrograman, dasar elektronika dan keterampilan matematika. (**Hernando Barragan, 2004**).

Banyak aplikasi sistem embedded yang secara alami di organisasikan menggunakan state machine. Suatu program harus merupakan aksi sekuensial atau sedang menangani suatu masukan ada baiknya di implementasikan menggunakan *state machine* (**Jerome Ding, 2000**)

Bagian dan suatu UML didasarkan pada konsep Staternachines, Aktivitas. dan lnteraksi. Statemachine membutuhkan aktivitas ketika menalankan suatu keadaan atau menjalankan transisi keadaan seperti “masukan”, “kerjakan”. “keluar’ atau pengaruh dan aktivitas yang mana pada gilirannya adalah terdiri atas suatu tindakan (**Tim dkk., 2005**)

Menurut **Anatoly S.** (2003) pemrograman berbasis state adalah merupakan suatu metoda perpindahan yang isomorphic dan formal dan transisi grafik kepada baris program.

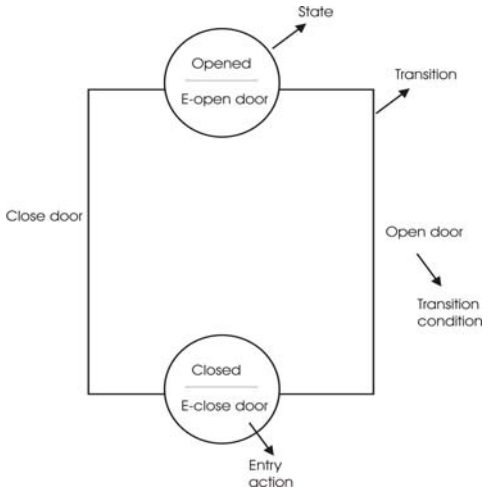
Sejak bahasa pemrograman berbasis visual menjadi sangat populer, secara tidak langsung maka baris perintah yang merupakan basis bahasa pemrograman tingkat tinggi adalah merupakan isu yang sangat penting (**Gergely dan Istvan, 2003**).

Berdasar pada tinjauan pustaka yang telah dikemukakan sebelumnya, maka akan dijelaskan dasar teori yang dibutuhkan untuk memecahkan masalah dalam penelitian ini. Dalam menulis baris perintah

suatu mikrokontroler, saat merancang sistem yang berbasis mikrokontroler, akan lebih mudah bila pertama kali dibuat langkah atau keadaan apa saja yang akan diterapkan pada sistem tersebut. Hal ini dapat menggunakan pendekatan *Finite State Machine* (FSM), karena menggunakan tabel transisi keadaan (*State Transition Table*) yang mampu menjelaskan hubungan langkah ataupun keadaan dan sistem yang dibangun tersebut.

2.1 Finite State Machine (FSM)

State Machine (FSM) adalah suatu metode pemrograman yang menggunakan "keadaan (state)". Keadaan pada kalimat tadi dapat berupa keadaan apa saja yang terjadi saat merancang dan menyusun langkah-langkah dalam membangun sistem. ini adalah pemodelan perilaku yang terdiri atas suatu keadaan, tindakan dan transisi. Status keadaan menyimpan informasi tentang masa lalu, yaitu mencerminkan perubahan masukan dan sistem yang dimulai pada masa lalu sampai kepada keadaan saat ini. Suatu transisi menandai adanya perubahan status keadaan dan transisi ini dijelaskan oleh suatu kondisi yang harus dipenuhi sampai transisi itu dimungkinkan untuk terjadi.



Gambar 1. Finite State Machine

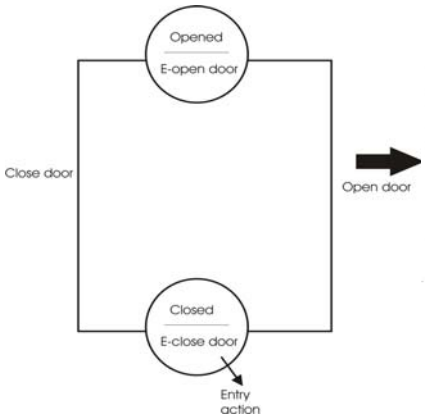
FSM dapat digambarkan dengan diagram keadaan (S*tate Chart Diagram*) atau state transition diagram seperti digambarkan pada Gambar 2.1. Disamping itu FSM juga dapat dibentuk dalam tabel keadaan transisi (S*tate Transition Table*). State transition table berupa tabel dua dimensi. Sisi vertikal menunjukkan keadaan yang
berlangsung saat ini, sedang sisi horisontal menunjukkan suatu keadaan selanjutnya. Perpotongan sisi vertikal dan horisontal merupakan Action (A) yang dilakukan sebagai akibat suatu Event (E).

Next / Current	S1	S2	.....	Sn
S1	-	**Ej/Ay**	.....	-
S2	-		.....	**Ei/Ax**
.....	.....	.....	.....	.....
Sn	**Ek/Az**	-	.....	-

(S : State, E: Event, A: Action, -:Illegal Transition)

Gambar 2. Contoh State Transition Table

Bila contoh Finite State Machine pada Gambar 2 di ubah ke dalam tabel keadaan transisi, seperti terlihat pada Gambar 3.



Open door

Next / Current	Closed	Opened
Opened	Open door/ Close_door	-
Closed	-	Close door/Open_door

Gambar 3. Perubahan FSM ke Tabel Keadaan Transisi

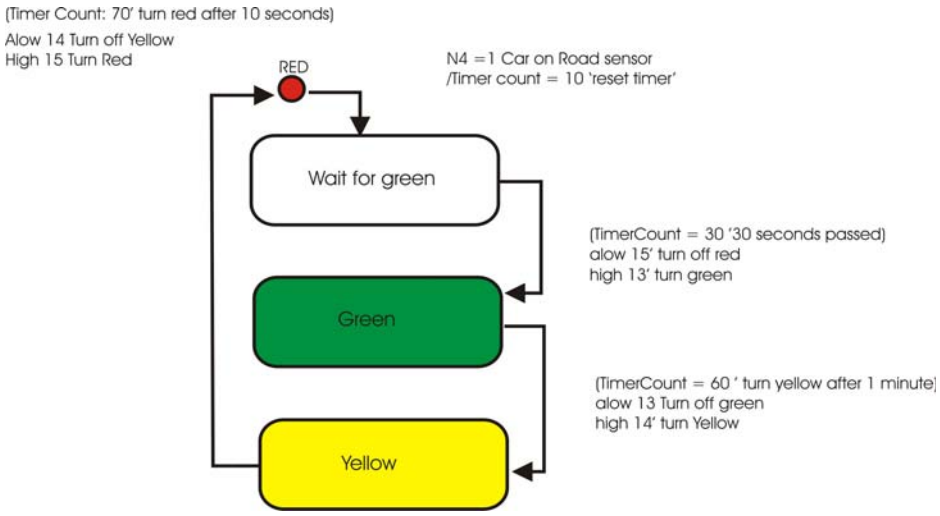
2.2 UML

Unified Modelling Language (UML) adalah suatu "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan suatu standar untuk merancang model suatu sistem. Dengan menggunakan UML dapat membuat model untuk semua jenis aplikasi piranti lunak, dengan aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan class dan operation dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek. Notasi UML terutama diturunkan dan 3 notasi yang telah ada sebelumnya:
Grady Booch OOD (Object-Oriented Design;**[desain berorientasi pada obyek]**), Jim Rumbaugh OMT (Object Modeling Technique;**[teknik pemodelan berbasis obyek]**). dan Ivar Jacobson OOSE (Object-Oriented Software Engineering; **[rekayasa perangkat lunak berorientasi obyek]**).

2.3 State Diagram

State Diagram menggambarkan transisi dan perubahan keadaan (dan satu state ke state lainnya) suatu objek pada sistem sebagai akibat dan stimuli yang diterima. Pada umumnya *State Diagram* menggambarkan class tertentu (satu class dapat memiliki lebih dan satu *State Diagram*).

Dalam UML, state digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar state umumnya memiliki kondisi G*uard* yang merupakan syarat terjadinya transisi yang bersangkutan. dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dan event tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

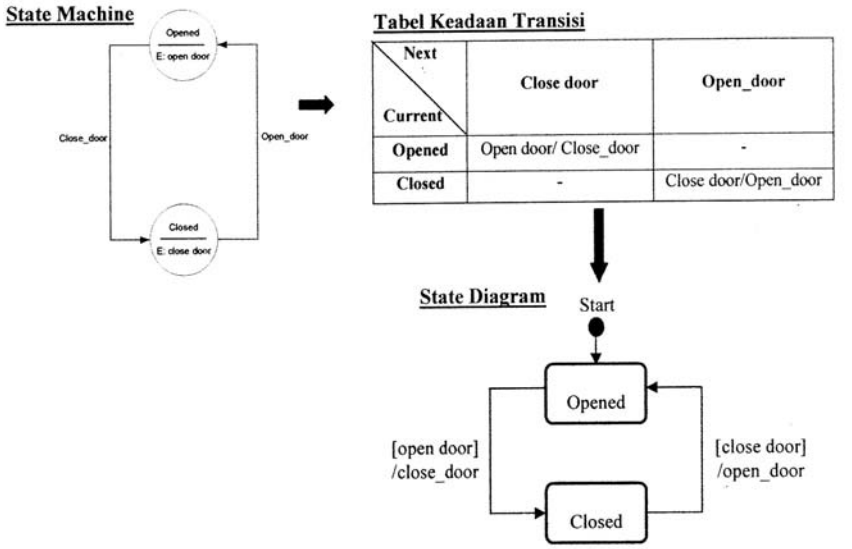


Gambar 4. Contoh State Diagram

State Diagram digunakan untuk menjelaskan perilaku suatu sistem. State diagram menguraikan semua keadaan yang mungkin dan suatu obyek sebagai suatu peristiwa. Setiap diagram pada umumnya menjelaskan obyek dan satu kelas (*class*) dan menjejaki keadaan dan setiap obyek di dalam satu sistem. Obyek merupakan kumpulan dan atribut yang diistilahkan sebagai keadaan, dan dapat berubah menjadi ke keadaan Iainnya melalui suatu transisi yang dipicu oleh suatu peristiwa.

2.4. Implementasi State Diagram menggunakan State Machine

Suatu sistem yang akan dirancang dapat di buat state machine dan tabel keadaan transisinya terlebih dahulu agar kita dapat mengetahui perilaku sistem. Baru
kemudian diterjemahkan ke dalam *State Diagram*. Agar Iebih jelasnya dapat dilihat pada berikut:



Gambar 5. Implementasi State Machine ke State Diagram

Pada Gambar 5 dapat dilihat bagaimana sistem untuk membuka dan menutup pintu dan digambarkan dengan *state machine*, dimana keadaan, kondisi transisi dan peristiwa yang terjadi setelah adanya kondisi transisi dipetakan dalam suatu tabel keadaan transisi. Dan dengan tabel keadaan transisi dapat dibentuk suatu gambar *State Diagram*. Penelitian ini akan membangun pembangkit baris perintah yang akan menerjemahkan S*tate Diagram* yang merupakan representasi dan state machine. Hasil terjemahan yang berupa baris perintah adalah merupakan ekstraksi dan *State Diagram* suatu sistem. Pada penelitian mi akan dilakukan pembuktian menggunakan simulasi pada sistem yang berbasis mikrokontroler. Dimana suatu mikrokontroler akan diprogram menggunakan *State Diagram*. Untuk kemudian diamati apakah hasil kerja mikrokontroler yang diprogram tadi telah sesuai dengan *State Diagram* yang dirancang.

2.5 Tahap-Tahap Penelitian

Ada beberapa tahapan yang akan dilakukan dalam menyelesaikan penelitian ini, yaitu:

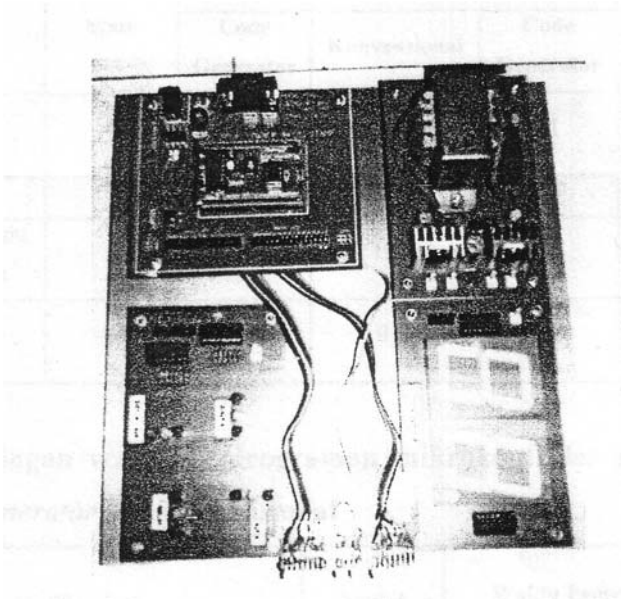
1. Mengumpulkan dan mempelajari bahan-bahan referensi/literature dari perpustakaan dan internet.
2. Menyusun kode program untuk *Code Generator*.
3. Membuat sistem untuk simulasi berbasis mikrokontroler Basic Stamp.
4. Melakukan uji coba program *Code Generator* untuk menterjemahkan *State Diagram* ke dalam baris perintah mikrokontroler Basic Stamp.
5. Melakukan uji coba dengan memuatkan hasil baris perintah yang dihasilkan oleh *Code Generator* kedalam sistem simulasi.
6. Menganalisa kesesuaian S*tate Diagram* dengan basil dan simulasi terkait pada baris perintah yang dihasilkan oleh *Code Generator*.
7. Menganalisa jumlah baris perintah yang dihasilkan oleh *Code Generator* dibanding dengan baris perintah yang diprogram secara langsung.
8. Menganalisa penggunaan memori mikrokontroler terkait dengan baris perintah yang dihasilkan oleh Code Generator dibanding dengan baris perintah yang diprogram secara langsung.

2.6 Hasil Penelitian

Sebelum menguji hasil dan program *Code Generator*, terlebih dahulu dibuat gambar *State Diagram* yang dibangun menggunakan program *WithClass*, program pendukung ini digunakan untuk menggambar *State Diagram* dan suatu sistem yang akan dibangun. Ada beberapa simulasi yang dibangun untuk menguji program *Code Generator ini.*

2.7 Simulasi

Perangkat simulasi yang dibangun ada beberapa bagian, yaitu sistem *downloader* Basic Stamp, simulasi sistem lampu lalu lintas, simulasi sistem pencacah dan simulasi kombinasi sistem lampu lalu lintas dan pencacah. sistem variasi lampu berjalan dengan pemilih dan pengatur gerak.



Gambar 6. Alat Simulasi *Code Generator* Untuk Mikrokontroler Basic Stamp

Dari simulasi yang telah dilakukan, maka ada beberapa analisa mengenai pemrograman dengan *Code Generator* dan pemrograman secara konvensional. Analisa yang pertama melakukan pengamatan pada penggunaan memori mikrokontroler dan jumlah baris perintah yang dihasilkan oleh *Code Generator* dibandingkan dengan baris perintah yang dibangun secara konvensional. Selanjutnya analisa yang kedua adalah menghitung lamanya waktu yang dibutuhkan dalam memprogram mikrokontroler baik dengan *Code Generator* maupun konvensional.

Tabel 1. Perbandingan pemrograman mikrokontroler menggunakan *Code Generator* dan konvensional

Nama Simulasi	Jumlah State	Penggunaan Memori		Baris Perintah	
		Code Generator	Konvensional	Code Generator	Konvensional
Simulasi Lampu Lalu-Lintas	4	8 %	4 %	71	51
Simulasi Pencacah	9	16 %	8 %	139	102
Simulasi Kombinasi Lampu Lalu-Lintas dan Pencacah	8	18 %	11 %	151	126
Simulasi Variasi LED Berjalan	6	15 %	9 %	101	71

Nama Simulasi	Jumlah State	Waktu Pemrograman	
		Code Generator	Konvensional
Simulasi Menampilkan Bilangan Genap antara 1 sampai dengan 20	4	37 menit	40 menit
Simulasi 7' segment hidup bergantian	2	10 menit	15 menit
Simulasi Menampilkan Bilangan Genap antara 1 sampai dengan 20	4	33 menit	35 menit
Simulasi LED Berjalan	4	20 menit	25 menit

2.8 Program *With Class*

Dalam penelitian ini program *Withclass* memberikan peranan yang cukup besar, karena dapat dengan mudah digunakan untuk menggambar S*tate Diagram* suatu sistem. Selain itu program ini juga menyediakan file *Interop. With_Class. dll* yang digunakan oleh C*ode Generator* untuk membuka tile S*tate Diagram* yang akan di ubah menjadi baris perintah mikrokontroler.

2.9 Program *Code Generator*

Perancangan *Code Generator* sangat membantu dalam pemrograman mikrokontroler Basic Stamp sehingga pemrograman menjadi

relatif lebih mudah. Dalam memprogram suatu sistem berbasis mikrokontroler cukup dengan menggambarkan *State Diagram*nya menggunakan program *With Clas*s, untuk kemudian di terjemahkan oleh *Code Generator*. Hal ini membuat perancangan sistem tersebut menjadi lebih baik, karena langkah-langkah keadaan suatu sistem dapat di gambarkan dalam *State Diagram* dan akhirnya di ubah ke dalam suatu baris perintah. Simulasi yang dibangun ada 4 jenis:

- Simulasi sistem lampu lalu lintas
- Simulasi sistem pencacah
- Simulasi kombinasi lampu lalu-lintas dan pencacah
- Simulasi variasi lampu berjalan dengan pemilih dan pengatur gerak

Pada ke-4 (empat) simulasi yang disebutkan diatas setelah di uji ternyata hasil simulasi telah sesuai dengan *State Diagram* dan sistem yang dirancang. Sistem yang dibangun pada saat simulasi adalah sistem yang sifatnya sekuensial, namun pada simulasi kombinasi lampu lalu-lintas dan pencacah di anggap sebagai sistem yang terkombinasi yaitu kombinasi antara sistem lampu lalu-lintas dan sistem pencacah. Setelah mewujudkan *State Diagram* dan sistem terkombinasi dan mensimulasikanya pada mikrokontroler, ternyata sistem terkombinasi tersebut merupakan sistem sekuensial.

2.10 Kinerja *Code Generator*

Kinerja dan *Code Generator* dapat dinilai dan seberapa mudah penggunaanya oleh para pemrogram mikrokontroler Basic Stamp pada khususnya. Pada pemrograman secara konvensional atau menggunakan *Code Generator* mempunyai tahapan pemrograman yang sedikit berbeda:

2.11 Pemrograman secara Konvensional

Pemrograman ini mempunyai tahapan sebagai berikut:

a. Analisa terhadap kebutuhan sistem yang dibangun
b. Membuat diagram alir dan sistem yang dibangun
c. Menulis baris perintah berdasarkan pada diagram alir dan sistem

2.12 Pemrograman dengan *Code Generator*

Pemrograman ini mempunyai tahapan sebagai berikut:

a. Analisa terhadap kebutuhan sistem yang dibangun

b. Membuat S*tate Diagram* dan sistem menggunakan program *WithClass* dan mensisipi baris perintah mikrokontroler pada S*tate Diagram*
c. Menggunakan program *Code Generator* untuk mengolah S*tate Diagram* menjadi baris perintah dan mikrokontroler

Berdasar pada tahapan yang ada pada pemrograman konvensional maupun pemrograman dengan *Code Generator* maka sebenarnya tidak ada perbedaan
Tahapan pada kedua model pemrograman. Hanya saja *Code Generator* memberi kemudahan dan keleluasaan dalam memprogram mikrokontroler, karena untuk memprogram sistem berbasis mikrokontroler cukup dengan menggambarkan *State Diagram* dan sistem yang akan dibangun. Selain itu waktu yang dibutuhkan untuk memprogram relatif lebih singkat bila menggunakan *Code Generator* di banding dengan cara konvensional.

3. Penutup

Kesimpulan yang dapat diambil dan penilitian yang dilakukan sampai saat mi adalah:

1. Tujuan penelitian untuk membangun sistem pemrograman mikrokontroler yang menggunakan *State Diagram* dan pengimplementasian bahasa berorientasi obyek pada mikrokontroler basic stamp telah tercapai.
2. Program *Withclass* sebagai program bantu sangat membantu dalam proses perancangan suatu *State Diagram*.
3. Interop. *WithClass*. dll sebagai suatu library yang sangat berguna dalam pembacaan gambar *State Diagram*.
4. Pemrograman mikrokontroler berbasis *State Diagram* lebih dapat menggambarkan bagaimana state didalam suatu sistem itu berinteraksi satu sama lainnya.
5. Pemrograman mikrokontroler berbasis *State Diagram* Iebih banyak menggunakan kapasitas memori dan menghasilkan baris perintah yang lebih panjang dibanding pemrograman secara konvensional.
6. Pemrograman menggunakan C*ode Generator* membutuhkan waktu relatif lebih
   singkat dibanding pemrograman secara konvensional.
7. Pemrograman mikrokontroler berbasis *State Diagram* sangat cocok untuk

memprogram suatu sistem yang sifatnya sekuensial.

Daftar Pustaka

_____ 2006. *BASIC Stamp Microcontrollers,*: http://www.parallax.com/

_____ 2006, *Developing Embedded Systems - A Tools Introduction*, http://microcontrollershop.com/An%2oEmbedded%2OTools%20lntroduction.php

_____ 2006, Develop embedded systems based on UML state machines, http://www.programmersheaven.com/2/

_____ 2006, Finite state machine, http://en.wikipedia.org/wiki/Finite state machine#Mathematical_model

_____ 2006. Microsojl .NET Framework Developer Centre. http://msdn.microsoft.com/netframeworkl

_____,2006, Netmedia Inc. Bas'cX by NetMedia Inc., http://www.basicx.com

_____ 2006, State diagram, http://en.wikipedia.org/wiki/State_diagram

_____ 2006, With Class, http://www.microgold.com/

Anatoly S., "Technology Of Automata-Based Programming", "PC World!Russia". 2003, No 10

B.Haberman and M. Trakhtenbrot, "An undergraduate program in embedded systems engineering," in 18th Conference on Software Engineering Education and Training), Apr.18—20, 2005, pp. 103—110.

B.Weiss, G. Gridling and M. Proske. "A Case Study in Efficient Microcontroller Education", University of Technology, Vienna (2003)

GenerExe, "Proramming State Machines On Microcontrollers Version 1.1"

Gergely P., Itsivan M., "Program Code Generation Based On UML Statechart Models", Periodica Polytechnica Ser. El. Eng. Vol. 47, No. 3-4, PP. 187-204 (2003)

Hemando B., "Wiring: Prototyping Physical Interaction Design", Interaction Design Institute lvrea (2004)

Parallax, Inc. "BASIC Stamp Programming Manual Version 1.9"

Sri D., Romi S.,'Pengantar Unified Modelling language (UML)". Kuliah Umum IlmuKomputer.Com Copyright © 2003 I ImuKomputer.Com

Tim dkk., "A Model-Based Approach for Executable Specifications on Reconfigurable Hardware", University of Paderborn/C-LAB Paderborn, Germany (2005)

W. Wolf and 1. Madsen, "Embedded svstetns education for the future," Proceedings of the IEEE, vol. 88, no. 1, pp. 23—30, Jan. 2000.