PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

ENGENHARIA DE COMPUTAÇÃO

PROJETO SINALEIRO INTELIGENTE

CURITIBA

2009

DENY LUCAS DA SILVA
RODRIGO KOTLEVSKI

PROJETO SINALEIRO INTELIGENTE

Este projeto será apresentado à disciplina de Problemas de Engenharia do Curso de Engenharia de Computação do Centro de Ciências Exatas e de Tecnologia da Pontifícia Universidade Católica do Paraná, como parte integrante da nota do segundo semestre.
Professor orientador: Afonso Ferreira Miguel.

CURITIBA
2009

SUMÁRIO

1.INTRODUÇÃO

Problemas de resolução de problemas em engenharia do curso de Engenharia de Computação, tem como intuito iniciar o desenvolvimento de projetos, desde a documentação completa,organogramas, cronogramas, apresentações e a conclusão do projeto nas mais corretas formas,já nos preparando para o mercado de trabalho mais a frente.

O grupo formado para o desenvolvimento do Projeto de resolução de problemas em engenharia do quarto período do curso de Engenharia de Computação é formado pelos seguintes integrantes: Deny Lucas da silva e Rodrigo Kotlevski.

A idéia do projeto surgiu em uma discussão entre os integrantes do grupo, sobre os cruzamentos em que nem existem movimento em uma via, e a outra via está supostamente cheia de carros e de como os sinaleiros demoram, por exemplo o sinaleiro que existe ali em garuva, a via da BR sempre está cheia, enquanto a via local passa algum carro de vez enquando. Então decidimos fazer um sinaleiro que resolvesse parcialmente este tipo de problema.

A partir deste momento, decidimos fazer um Sinaleiro Inteligente.

1.1 JUSTIFICATIVAS

O projeto “Sinaleiro Inteligente“,se destaca pelo tempo em que ele vai economizar na via, e assim trazendo melhor fluxo nas vias ,trazendo melhor bem estar entre as pessoas diminuindo o stress no trânsito.

Com o crescimento amplo da tecnologia, podemos cada vez mais melhorar tanto os nosso circuitos, quanto conseguindo colocar essas tecnologias novas para o melhorio da nossas vidas trazendo assim um melhor bem-estar.

1.2 METODOLOGIA

O projeto sinaleiro inteligente foi gerado para que nos conseguíssemos aplicar ele no nosso dia-a-dia. Fizemos uma pesquisa para saber qual o melhor jeito de implementarmos o projeto, mas encontramos muitas dificuldades em fazer uma união entre o circuitos, a programação e o sensor. Tivemos que pensar bastante de como íamos fazer o link entre os três.

Nas primeiras semanas começamos a enrolar as bobinas, depois as placas para o sensor com a ajuda do osciloscópio para obtermos os sinais, fizemos a maquete, depois a programação e por último tentamos interligar os três para obter o funcionamento.

1.3 AS RESPONSABILIDADES

Para que o projeto obtivesse susseso em seu desenvolvimento foi necessário a participação ativa de todos os participantes do grupo e também dos professores, é necessário muita responsabilidade, seriedade e muita força de vontade em todos os eixos do grupo para que o projeto fosse bem desenvolvido.Cada integrante teve a sua responsabilidade e cumpriu com o máximo de comprometimento para com ele. Os professores estavam aptos a responder todas nossas dúvidas em relação ao projeto, e nos ajudar, dar novas idéias e apoio. E também dependemos das estruturas da PUC, que se tornou a principal responsabilidade, pois são nos laboratórios e os técnicos que estão nesses para nos dar uma assistência que conseguíamos fazer os devidos testes.

2. OS OBJETIVOS

O objetivo do sinaleiro inteligente, “ensinar” o grupo a fazer pesquisas, documentações, cronogramas, apresentações, ou seja, tudo que envolve um bom gerenciamento de projetos, além do trabalho com placas, circuitos, osciloscópio, assim envolvendo a matéria vista em sala de aula para a nossa prática, como as aulas de sistemas digitais nos envolvendo na matéria, circuitos elétricos nos envolvendo nos circuitos, Física envolvendo bobinas. E de terminarmos esse projeto para nos desenvolvermos no conhecimento no andamento do curso.

3. NÃO ESTÁ INCLUSO NO ESCOPO DESTE PROJETO

O que não foi incluso no projeto, mas quer poderia ser implementado, era o sinaleiro para pedestre, pois enquanto o sinaleiro ficasse em vermelho, o sinaleiro de pedestre fosse abrir para os pedestres fluírem na via.

4. O PROJETO

O projeto consiste em fazer uma melhoria nos sinaleiros do trânsito, no qual o movimento dos carros são contados por uma bobina, em que esta bobina iria se comunicar com um circuito para mandar o números de carros que estão passando na via, e assim comandado pelo computador, o melhor tempo para o sinaleiro.

Na primeira etapa, a bobina vai sentir o campo de um metal passando por cima dele, e vai envia a informação para a placa.

Na segunda etapa, este circuito vai se comunicar com o altera para que saia a contagem.

Na terceira etapa, ele vai se comunicar com o circuito RS 232 que ira se comunicar com a porta serial e que com a programação, vai mudar a luz do programa em que esta rodando, assim fazendo as cores muda.

5.OS COMPONENTES UTILIZADOS

2 pic 12F675;

3 2 78L05;

4 2 capacitoes 10 n F;

5 2 capacitores 100 n F;

6 2 bobinas fio numero 28;

7 4 capacitores 1 micro F;

8 1 circuito RS 232;

9 2 pic 12F683;

10 Resistores;

11 2 indutor 330 micro H;

6. OS RESULTADOS

Não conseguimos concluir com êxito totalmente o projeto. Ao longo do tempo, fomos ver que nossos conhecimentos não foram hábeis para alcançar o objetivo com êxito.

Conseguimos desenvolver ele quase que por inteiro, mas na hora de combinar o circuito com o software, para terminarmos ele completamente, obtemos problemas que não sabíamos resolver.

O projeto porem foi bastante gratificante em fazê-lo, mesmo não obtendo êxito, pelos problemas que obtivemos, mesmo assim vamos entregar:

Software de controle em C++;

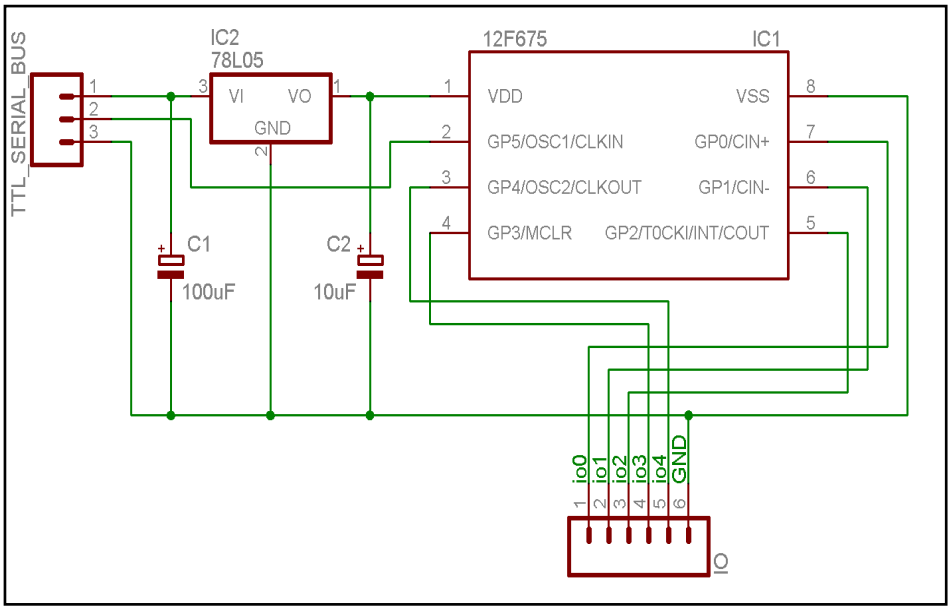
CD com arquivos, fotos, desenhos, códigos-fonte, esquemáticos, diagramas e modelos dos módulos implementados;

Vídeo;

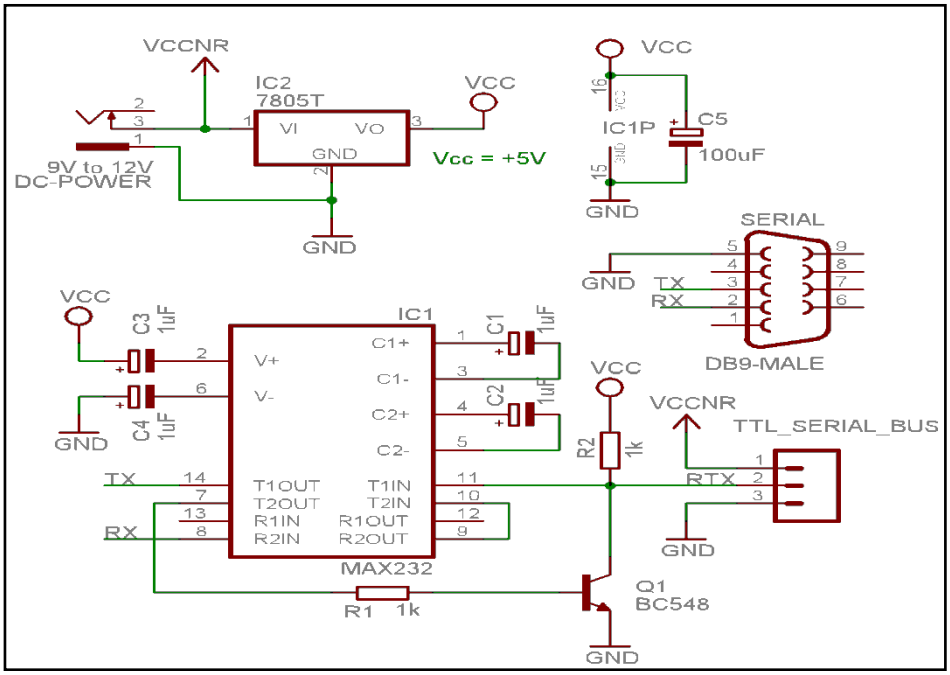
Documentação do projeto dos itens acima.

7.Esquemático dos Circuitos

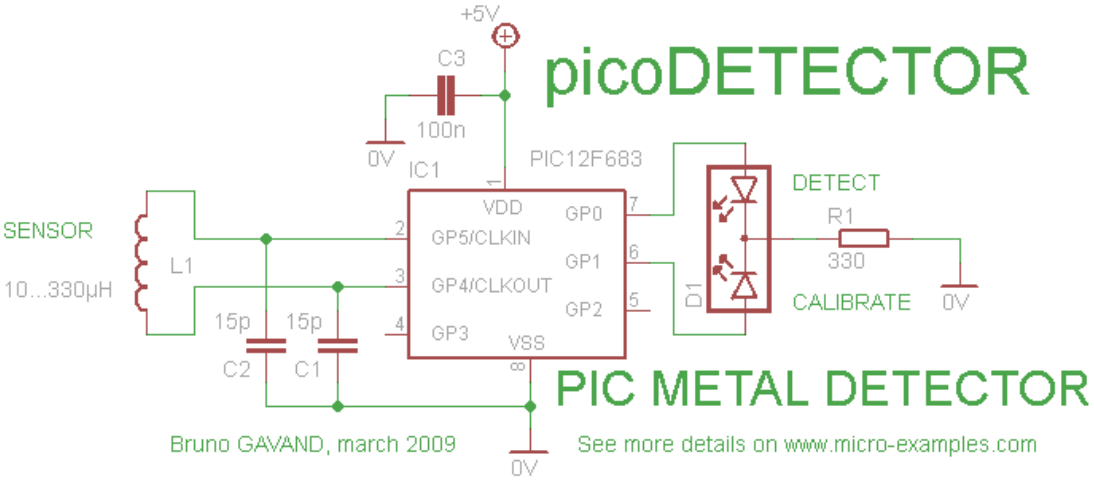
Nós utilizamos dois módulos M2.



RS232



Pico detector de metais foram 2



Programação pico detector

```
#define MAXTRY 15       // number of watchdog restart to calibrate loo
p counter

unsigned char   ctr ;           // number of loops between two watchdo
g resets
unsigned char   previous ;      // previous value of ctr
unsigned char   calibr ;        // calibration value when oscillator r
uns free
unsigned char   restarts ;      // number of watchdog restarts
unsigned char   en ;            // enable flag, allows detection

/*
 * main loop
 */
void    main()
        {
        unsigned char   i ;


        /*
         * configure GPIO as digital port
         */
        CMCON0 = 7 ;
        ANSEL = 0 ;
        TRISIO = 0 ;
        GPIO = 0 ;

        /*
         * power up ?
         */
        if(STATUS.NOT_TO)
                {
                /*
                 * yes, init variables
                 */
                restarts = 0 ;
                calibr = 1 ;
                }

        /*
         * watchdog reset counter
         */
        if(restarts < 255) restarts++ ;

        /*
         * if counter differs too much from calibration value
         */
        if((previous ^ ctr) > calibr)
                {
                /*
                 * turn detect LED on
                 */
                GPIO.F0 = en ;

                /*
                 * if not on power up
                 */
                if(STATUS.NOT_TO == 0)
```

```
                {
                /*
                 * while in calibration mode
                 */
                if(restarts < MAXTRY)
                        {
                        /*
                         * shift calibration value
                         * and wait a little bit
                         */
                        calibr <<= 1 ;
                        Delay_ms(5) ;
                        }
                }
        else
                {
                /*
                 * turn detect LED off
                 */
                GPIO.F0 = 0 ;
                }
        }

/*
 * save last counter
 */
previous = ctr ;

/*
 * is calibration over
 */
if(restarts > MAXTRY)
        {
        /*
         * yes, turn calibrate LED off
         * and set enable flag
         */
        GPIO.F1 = 0 ;
        en = 1 ;
        }
else
        {
        /*
         * no, turn calibrate LED on
         * and clear enable flag
         */
        GPIO.F1 = 1 ;
        en = 0 ;
        }

/*
 * set watchdog prescaler
 */
OPTION_REG = 0b11111001 ;

/*
 * start counter, to be interrupted by watchdog
 */
ctr = 0 ;
for(;;)
       {
```

```
            ctr++ ;
            }
    }
```

Programação projeto

Lampada.h

```
#pragma once

#include "stdafx.h"
#include <iostream>

using namespace std;
//Definição da classe Lampada.

class Lampada
{

public:
      enum Estados
      {
            apagada, ligada, queimada, //define os campos que terá a
variável do tipo Estados que será utilizada: define o estado
individual de cada lampada.
      };
      Estados estado; //cria uma variavel do tipo Estados com o fim de
controlar o "estado" de cada lampada.
public:
      Lampada(void); //contrutor default da classe lampada
      ~Lampada(void); //destrutor da classe lampada
      void liga(void); //método liga
      void desliga(void); //método desliga
      Estados ver_estado(void); //método que verifica o estado da
lampada (apagada, ligada ou queimada)
      void queima(void); //método que queima a lampada selecionada

      };
```

Lampada.cpp

```
#include "stdafx.h"
#include "Lampada.h"

Lampada::Lampada(void)
{
	estado = apagada; //contrutor default: inicializa a lampada no
estado "apagada".
}

Lampada::~Lampada(void) //destrutor
{
}

void Lampada::liga(void)
{
	estado = ligada; //atribui o estado "ligada" a lampada.
}

void Lampada::desliga(void)
{
	estado = apagada; //atribui o estado "apagada" a lampada.
}

void Lampada::queima(void)
{
	estado = queimada; //atribui o estado "queimada" a lampada.
}

Lampada::Estados Lampada::ver_estado(void) //verifica o estado da
lampada (ligada, apagada ou queimada)
{
	if(estado == apagada)
	{
		return apagada; //se a lampada estiver apagada, o método
retorna o valor equivalente ao estado "apagada".
	}

	else if(estado == ligada)
		{
	    return ligada; //se a lampada estiver ligada, o método
retorna o valor equivalente ao estado "ligada".
	}

	else
	{
		return queimada; //se a lampada estiver queimada, o método
retorna o valor equivalente ao estado "queimada".
	}

	}
```

Wait.h

```
#include "stdafx.h"
#include <time.h>
//using namespace std;
#pragma once

class Wait
{
public:
	Wait(void); //contrutor default da classe Wait.
	~Wait(void); //destrutor.
	void wait (int seconds); //método que faz com que o inteiro
passado por parametro seja utilizado como tempo de espera.
	};
```

Wait.cpp

```
Wait::Wait(void)
{
}

Wait::~Wait(void)
{
}

void Wait::wait ( int seconds )
{
  clock_t endwait;
  endwait = clock () + seconds * CLOCKS_PER_SEC;
  while (clock() < endwait) {}
}
```

Stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
#pragma once

	// TODO: reference additional headers your program requires here
```

Stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
// semaforo.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

	#include "stdafx.h"
```

Assembly info.cpp

```
#include "stdafx.h"

using namespace System;
using namespace System::Reflection;
using namespace System::Runtime::CompilerServices;
using namespace System::Runtime::InteropServices;
using namespace System::Security::Permissions;

//
// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
//
[assembly:AssemblyTitleAttribute("semaforo")];
[assembly:AssemblyDescriptionAttribute("")];
[assembly:AssemblyConfigurationAttribute("")];
[assembly:AssemblyCompanyAttribute("")];
[assembly:AssemblyProductAttribute("semaforo")];
[assembly:AssemblyCopyrightAttribute("Copyright (c)  2009")];
[assembly:AssemblyTrademarkAttribute("")];
[assembly:AssemblyCultureAttribute("")];

//
// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the value or you can default the Revision and
Build Numbers
// by using the '*' as shown below:

[assembly:AssemblyVersionAttribute("1.0.*")];

[assembly:ComVisible(false)];

[assembly:CLSCompliantAttribute(true)];

[assembly:SecurityPermission(SecurityAction::RequestMinimum,
UnmanagedCode = true)];
```

Semaforo.h

```
#pragma once

#include "stdafx.h"
#include "Lampada.h"
//Definição da classe Semaforo (carros).
class Semaforo
{
public:
	enum Estados
	{
		verm, ama, verd, desligado,   //define os campos que terá a
variável do tipo Estados que será utilizada: define uma fase
individual para cada semaforo.
	};

public:
	class Lampada vermelha, amarela, verde; //cria 3 objetos da
classe Lampada.
	int time, time_verm, time_ama, time_verde; //cria variaveis do
tipo int para manipular o tempo de cada lampada e o tempo geral
decorridos.
	Estados estado; //cria uma variavel do tipo Estados para
controlar o estado (fase) do semáforo.

public:
	Semaforo(void); //contrutor default da classe Semaforo.
	Semaforo(int, int, int); //contrutor parametrizado da classe
Semaforo.
	~Semaforo(void); //destrutor
	void liga(Estados); //método liga
	void desliga(void); //método desliga
	void queima_sem(void); //método que queima uma lampada do
semaforo previamente selecionada pelo usuário.
	void tick(void); //método que controla a passagem do tempo.
	void show(void); //método que exibe a situação do semaforo.
	void set_estado(Estados); //método para alterar o estado do
semaforo (vermelho, amarelo, verde ou desligado).
```

Semaforo.cpp

```
#include "stdafx.h"
#include "Semaforo.h"

Semaforo::Semaforo(void) //construtor default da classe Semaforo:
inicializa o timer de cada lampada.
{
	time_verm = 5; //atribui o tempo de 5 segundos à lampada
vermelha.
	time_ama = 2; //atribui o tempo de 2 segundos à lampada amarela.
	time_verde = 3; //atribui o tempo de 3 segundos à lampada verde.
	time = 0; //atribui zero ao contador de tempo geral (auxiliar).

	this->desliga(); //inicializa o semaforo desligado, ou seja, com
todas as lampadas apagadas.
}

Semaforo::~Semaforo(void) //destrutor
```

```
{
}
void Semaforo::liga(Estados fase)
{
	time = 0; //inicializa o timer zerado.

	if(fase == verm) //se a lampada escolhida para ser ligada for a
vermelha, liga lampada vermelha desligando as outras.
	{
		set_estado(verm); //altera o estado do semaforo para "verm"
(vermelho).

		if(vermelha.ver_estado() != Lampada::queimada) //Nao
permite que a lampada mude de estado caso esteja queimada.
			vermelha.liga(); //liga a lampada vermelha.

		if(amarela.ver_estado() != Lampada::queimada) //Nao permite
que a lampada mude de estado caso esteja queimada.
			amarela.desliga(); //desliga a lampada amarela.

		if(verde.ver_estado() != Lampada::queimada) //Nao permite
que a lampada mude de estado caso esteja queimada.
			verde.desliga(); //desliga a lampada verde.
	}

	if(fase == ama) //se a lampada escolhida para ser ligada for a
amarela, liga lampada amarela desligando as outras.
	{
		set_estado(ama); //altera o estado do semaforo para "ama"
(amarelo).

		if(vermelha.ver_estado() != Lampada::queimada) //Nao
permite que a lampada mude de estado caso esteja queimada.
			vermelha.desliga(); //desliga a lampada vermelha.

		if(amarela.ver_estado() != Lampada::queimada) //Nao permite
que a lampada mude de estado caso esteja queimada.
			amarela.liga(); //liga a lampada amarela.

		if(verde.ver_estado() != Lampada::queimada) //Nao permite
que a lampada mude de estado caso esteja queimada.
			verde.desliga(); //desliga a lampada verde.
	}

	if(fase == verd) //se a lampada escolhida para ser ligada for a
verde, liga lampada verde desligando as outras.
	{
		set_estado(verd); //altera o estado do semaforo para "verd"
(verde).

		if(vermelha.ver_estado() != Lampada::queimada) //Nao
permite que a lampada mude de estado caso esteja queimada.
			vermelha.desliga(); //desliga a lampada vermelha.

		if(amarela.ver_estado() != Lampada::queimada) //Nao permite
que a lampada mude de estado caso esteja queimada.
			amarela.desliga(); //desliga a lampada amarela.

		if(verde.ver_estado() != Lampada::queimada) //Nao permite
que a lampada mude de estado caso esteja queimada.
			verde.liga(); //liga a lampada verde.
```

```
	}
}

void Semaforo::desliga(void)
{
	set_estado(desligado); //altera o estado do semaforo para
"desligado".

	if(vermelha.ver_estado() != Lampada::queimada) //Nao permite que
a lampada mude de estado caso esteja queimada.
		vermelha.desliga(); //desliga a lampada vermelha.

	if(amarela.ver_estado() != Lampada::queimada) //Nao permite que
a lampada mude de estado caso esteja queimada.
		amarela.desliga(); //desliga a lampada amarela.

	if(verde.ver_estado() != Lampada::queimada) //Nao permite que a
lampada mude de estado caso esteja queimada.
		verde.desliga(); //desliga a lampada verde.
}
void Semaforo::queima_sem(void)
{
	int opcao;

	cout << endl;
    //Exibe um pequeno menu para que o usuario possa escolher a
lampada a ser queimada:
	do
	{
		cout << "Queimar lampada:" << endl;
		cout << "\t 1) Vermelha" << endl;
		cout << "\t 2) Amarela" << endl;
		cout << "\t 3) Verde" << endl;

		cin >> opcao;
	}while(opcao != 1 && opcao != 2 && opcao != 3);

	if(opcao == 1) //se a opção escolhida for a "1", queima a
lampada vermelha.
		vermelha.queima();
	if(opcao == 2) //se a opção escolhida for a "2", queima a
lampada amarela.
		amarela.queima();
	if(opcao == 3) //se a opção escolhida for a "3", queima a
lampada verde.
		verde.queima();
}
void Semaforo::tick(void)
{
	if(time == time_verm && estado == verm)//se o contador geral
atingiu o tempo da lampada vermelha e o semaforo estiver na fase
vermelha
		liga(verd);			//entao a lampada verde
é acionada.
	if(time == time_ama && estado == ama) //se o contador geral
atingiu o tempo da lampada amarela e o semaforo estiver na fase
amarela
		liga(verm);			//entao a lampada
vermelha é acionada.
	if(time == time_verde && estado == verd)//se o contador geral
atingiu o tempo da lampada verde e o semaforo estiver na fase verde
```

```
            liga(ama);                          //entao a lampada amarela
é acionada.

      time++;      //contador geral (auxiliar).
}

void Semaforo::show(void)
{   //verifica em qual estado está a lampada vermelha:
      if(vermelha.ver_estado() == Lampada::ligada)//se estiver ligada,
imprime "Vermelho".
            cout << "Vermelho" << endl;
      else if(vermelha.ver_estado() == Lampada::apagada)//se estiver
apagada, imprime "0".
            cout << "0" << endl;
      else //se nao estiver ligada nem apagada é porque está queimada,
entao imprime "Queimada".
            cout << "Queimada" << endl;

    //verifica em qual estado está a lampada amarela:
      if(amarela.ver_estado() == Lampada::ligada) //se estiver ligada,
imprime "Amarelo".
            cout << "Amarelo" << endl;
      else if(amarela.ver_estado() == Lampada::apagada)//se estiver
apagada, imprime "0".
            cout << "0" << endl;
      else //se nao estiver ligada nem apagada é porque está queimada,
entao imprime "Queimada".
            cout << "Queimada" << endl;

    //verifica em qual estado está a lampada verde:
      if(verde.ver_estado() == Lampada::ligada)//se estiver ligada,
imprime "Verde".
            cout << "Verde" << endl;
      else if(verde.ver_estado() == Lampada::apagada)//se estiver
apagada, imprime "0".
            cout << "0" << endl;
      else //se nao estiver ligada nem apagada é porque está queimada,
entao imprime "Queimada".
            cout << "Queimada" << endl;
}
void Semaforo::set_estado(Estados estado)
{
      this->estado = estado; //atribui o novo estado ao estado
original.
}
```

Semaforos.cpp

```
#include "stdafx.h"
#include "Form1.h"
#include "Cruzamento.h"
#include "Wait.h"

using namespace serial;


[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
	// Enabling Windows XP visual effects before any controls are
created
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	// Create the main window and run it
	Application::Run(gcnew Form1());
	return 0;
	}
```

Cruzamento.h

```
#include "Semaforo.h"
#include "stdafx.h"
//using namespace std;
#pragma once
//Definição da classe cruzamento.
class Cruzamento
{
public:
	class Semaforo semaforo1, semaforo2; //cria 2 objetos da classe
Semaforo.
public:
	Cruzamento(void); //contrutor default da classe Cruzamento.
	~Cruzamento(void); //destrutor
	void liga_cruz(void); //método que liga o cruzamento em um
estado estável pre-definido.
	void tick_cruz(void); //método que controla a passagem do tempo
no cruzamento.
	void show_cruz(void); //método que exibe a situação atual do
cruzamento.
	void queima_cruz(void); //método que permite ao usuário escolher
qual lampada deseja queimar em qual semáforo.

	};
```

Cruzamento.cpp

```
#include "stdafx.h"
#include "Cruzamento.h"
//using namespace std;

Cruzamento::Cruzamento(void) //contrutor default da classe Cruzamento.
{

}

Cruzamento::~Cruzamento(void) //destrutor.
{
}
void Cruzamento::liga_cruz(void)
{ //Inicializa o cruzamento em um estado estável pre-definido:
      semaforo1.liga(Semaforo::verd); //liga o primeiro semaforo de
carros na fase "verd" (verde).
      semaforo2.liga(Semaforo::verm); //liga o segundo semaforo de
carros na fase "verm" (vermelho).
}

void Cruzamento::tick_cruz(void)
{
      semaforo1.tick(); //faz a chamada do método que controla o tempo
para o primeiro semaforo de carros.
      semaforo2.tick(); //faz a chamada do método que controla o tempo
para o segundo semaforo de carros.
}
void Cruzamento::show_cruz(void)
{   //Imprime na tela a situação geral dos 4 conjuntos de lampadas:
      cout << "Semaforo 1:" << endl;
      semaforo1.show(); //imprime na tela a situação do primeiro
semaforo de carros.
      cout << endl << endl << "Semaforo 2:" << endl;
      semaforo2.show(); //imprime na tela a situação do segundo
semaforo de carros.
}
void Cruzamento::queima_cruz(void)
{
      int opcao;

      system("cls");
    //Exibe um pequeno menu para que o usuario possa escolher em qual
semaforo deseja queimar uma lampada:
      do
      {
            cout << "Queimar lampada do semaforo:" << endl;
            cout << "\t 1) Primeiro Semaforo" << endl;
            cout << "\t 2) Segundo Semaforo" << endl;

            cin >> opcao;
      }while(opcao != 1 && opcao != 2 && opcao != 3 && opcao != 4);

      if(opcao == 1) //se a opção escolhida for a "1", queimará uma
lampada do primeiro semaforo de carros.
            semaforo1.queima_sem();
      if(opcao == 2) //se a opção escolhida for a "2", queimará uma
lampada do segundo semaforo de carros.
            semaforo2.queima_sem();
}
```

Form.h

```
#pragma once

#include "Semaforo.h"
#include "Cruzamento.h"
#include "Wait.h"
#include "Lampada.h"

      Cruzamento cruzamento;
      Wait wait;

namespace serial {

      using namespace System;
      using namespace System::ComponentModel;
      using namespace System::Collections;
      using namespace System::Windows::Forms;
      using namespace System::Data;
      using namespace System::Drawing;
      using namespace std;

      /// <summary>
      /// Summary for Form1
      ///
      /// WARNING: If you change the name of this class, you will need
to change the
      ///          'Resource File Name' property for the managed
resource compiler tool
      ///          associated with all .resx files this class depends
on.  Otherwise,
      ///          the designers will not be able to interact properly
with localized
      ///          resources associated with this form.
      /// </summary>
      public ref class Form1 : public System::Windows::Forms::Form
      {
      public:

      //          property Keys KeyCode {
  //  Keys get ();
//}
        int x;

            Form1(void)
            {
                  InitializeComponent();

//             serialPort1->Open();
                  x = 0;

            //
                  //TODO: Add the constructor code here
                  //
            }

      protected:
            /// <summary>
```

```
            /// Clean up any resources being used.
            /// </summary>
            ~Form1()
            {

//                  serialPort1->Close();
                  if (components)
                  {
                        delete components;

                  }
            }

      protected:

      protected:

      private: System::Windows::Forms::GroupBox^  groupBox1;
      internal: System::Windows::Forms::RadioButton^  verde_aceso;
      private:
      internal: System::Windows::Forms::RadioButton^  amarelo_aceso;
      internal: System::Windows::Forms::RadioButton^  vermelho_aceso;
      private: System::Windows::Forms::CheckBox^  checkBox1;
      private: System::Windows::Forms::GroupBox^  groupBox2;
      internal: System::Windows::Forms::RadioButton^  radioButton1;
      private:
      internal: System::Windows::Forms::RadioButton^  radioButton2;
      internal: System::Windows::Forms::RadioButton^  radioButton3;
      internal: System::Windows::Forms::RadioButton^  radioButton4;
      internal: System::Windows::Forms::RadioButton^  radioButton5;
      internal: System::Windows::Forms::RadioButton^  radioButton6;
      private: System::Windows::Forms::Label^  label1;
      private: System::IO::Ports::SerialPort^  serialPort1;
      private: System::ComponentModel::IContainer^  components;
      private:
            /// <summary>
            /// Required designer variable.
            /// </summary>


#pragma region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            void InitializeComponent(void)
            {
                  this->components = (gcnew
System::ComponentModel::Container());
                  this->groupBox1 = (gcnew
System::Windows::Forms::GroupBox());
                  this->verde_aceso = (gcnew
System::Windows::Forms::RadioButton());
                  this->amarelo_aceso = (gcnew
System::Windows::Forms::RadioButton());
                  this->vermelho_aceso = (gcnew
System::Windows::Forms::RadioButton());
                  this->checkBox1 = (gcnew
System::Windows::Forms::CheckBox());
                  this->groupBox2 = (gcnew
System::Windows::Forms::GroupBox());
```

```
                    this->radioButton1 = (gcnew
System::Windows::Forms::RadioButton());
                    this->radioButton2 = (gcnew
System::Windows::Forms::RadioButton());
                    this->radioButton3 = (gcnew
System::Windows::Forms::RadioButton());
                    this->radioButton4 = (gcnew
System::Windows::Forms::RadioButton());
                    this->radioButton5 = (gcnew
System::Windows::Forms::RadioButton());
                    this->radioButton6 = (gcnew
System::Windows::Forms::RadioButton());
                    this->label1 = (gcnew
System::Windows::Forms::Label());
                    this->serialPort1 = (gcnew
System::IO::Ports::SerialPort(this->components));
                    this->groupBox1->SuspendLayout();
                    this->groupBox2->SuspendLayout();
                    this->SuspendLayout();
                    //
                    // groupBox1
                    //
                    this->groupBox1->Controls->Add(this->verde_aceso);
                    this->groupBox1->Controls->Add(this->amarelo_aceso);
                    this->groupBox1->Controls->Add(this->vermelho_aceso);
                    this->groupBox1->Location =
System::Drawing::Point(16, 15);
                    this->groupBox1->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->groupBox1->Name = L"groupBox1";
                    this->groupBox1->Padding =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->groupBox1->Size = System::Drawing::Size(467,
295);
                    this->groupBox1->TabIndex = 0;
                    this->groupBox1->TabStop = false;
                    this->groupBox1->Text = L"Cruzamento 1";
                    //
                    // verde_aceso
                    //
                    this->verde_aceso->AutoCheck = false;
                    this->verde_aceso->AutoSize = true;
                    this->verde_aceso->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->verde_aceso->ForeColor =
System::Drawing::Color::Lime;
                    this->verde_aceso->Location =
System::Drawing::Point(9, 98);
                    this->verde_aceso->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->verde_aceso->Name = L"verde_aceso";
                    this->verde_aceso->Size = System::Drawing::Size(86,
29);
                    this->verde_aceso->TabIndex = 0;
                    this->verde_aceso->TabStop = true;
                    this->verde_aceso->Text = L"Verde";
                    this->verde_aceso->UseVisualStyleBackColor = true;
                    //
```

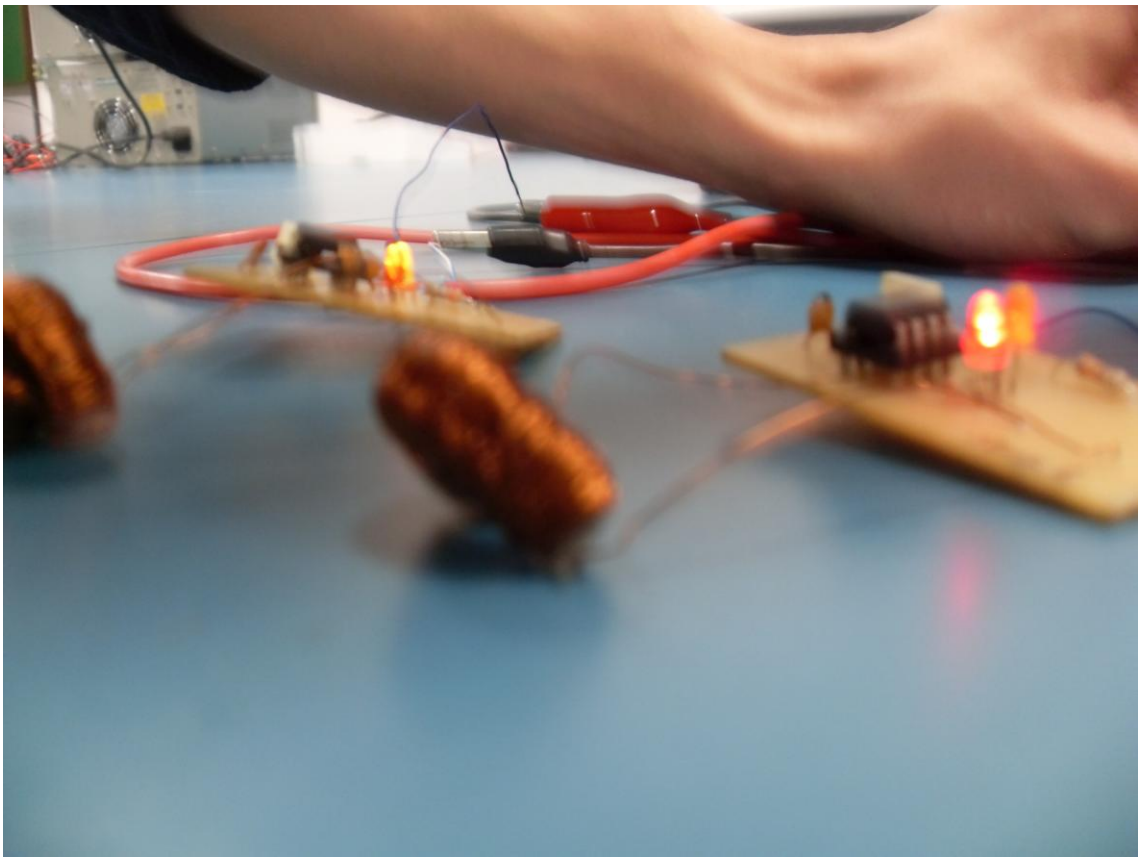
```
                    // amarelo_aceso
                    //
                    this->amarelo_aceso->AutoCheck = false;
                    this->amarelo_aceso->AutoSize = true;
                    this->amarelo_aceso->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->amarelo_aceso->ForeColor =
System::Drawing::Color::Yellow;
                    this->amarelo_aceso->Location =
System::Drawing::Point(8, 62);
                    this->amarelo_aceso->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->amarelo_aceso->Name = L"amarelo_aceso";
                    this->amarelo_aceso->Size =
System::Drawing::Size(106, 29);
                    this->amarelo_aceso->TabIndex = 0;
                    this->amarelo_aceso->TabStop = true;
                    this->amarelo_aceso->Text = L"Amarelo";
                    this->amarelo_aceso->UseVisualStyleBackColor = true;
                    //
                    // vermelho_aceso
                    //
                    this->vermelho_aceso->AutoCheck = false;
                    this->vermelho_aceso->AutoSize = true;
                    this->vermelho_aceso->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->vermelho_aceso->ForeColor =
System::Drawing::Color::Red;
                    this->vermelho_aceso->Location =
System::Drawing::Point(9, 25);
                    this->vermelho_aceso->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->vermelho_aceso->Name = L"vermelho_aceso";
                    this->vermelho_aceso->Size =
System::Drawing::Size(117, 29);
                    this->vermelho_aceso->TabIndex = 0;
                    this->vermelho_aceso->TabStop = true;
                    this->vermelho_aceso->Text = L"Vermelho";
                    this->vermelho_aceso->UseVisualStyleBackColor = true;
                    this->vermelho_aceso->CheckedChanged += gcnew
System::EventHandler(this, &Form1::vermelho_aceso_CheckedChanged);
                    //
                    // checkBox1
                    //
                    this->checkBox1->AutoSize = true;
                    this->checkBox1->Location =
System::Drawing::Point(16, 338);
                    this->checkBox1->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->checkBox1->Name = L"checkBox1";
                    this->checkBox1->Size = System::Drawing::Size(149,
21);
                    this->checkBox1->TabIndex = 1;
                    this->checkBox1->Text = L"Sinaleiros ligados\?";
                    this->checkBox1->UseVisualStyleBackColor = true;
```

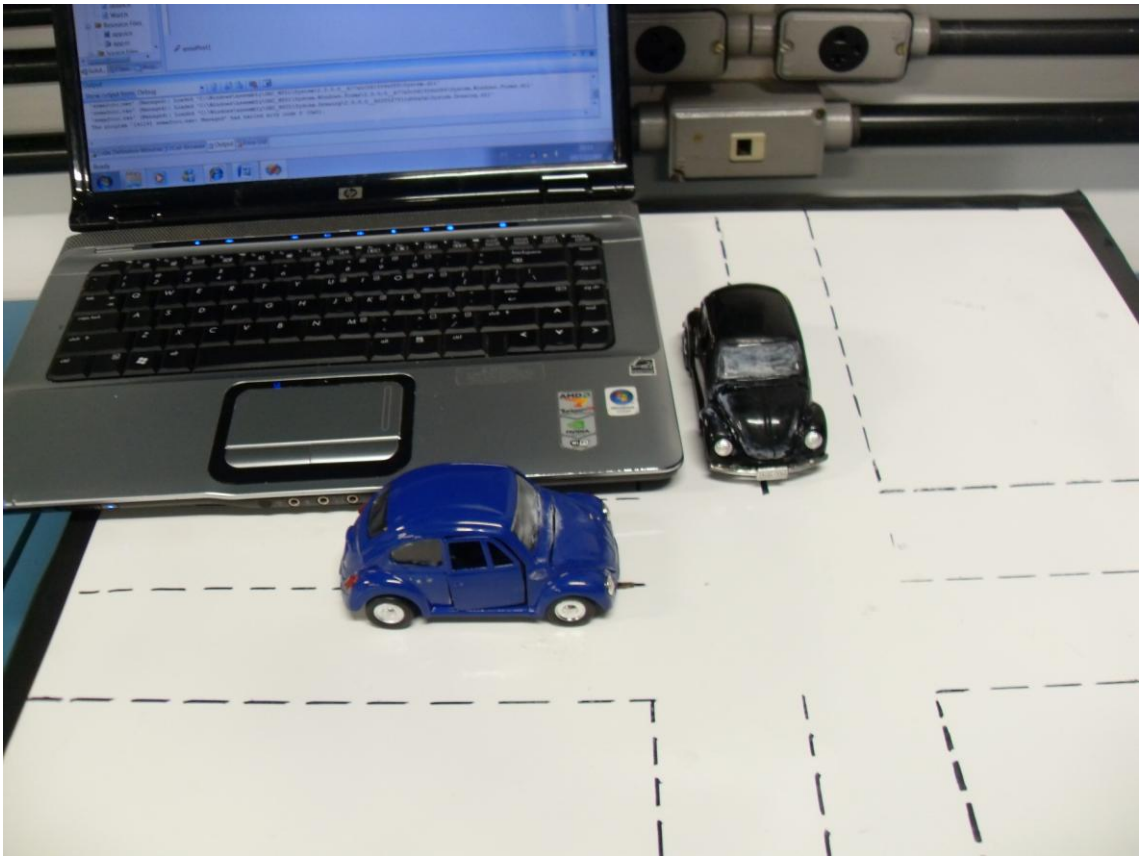
```
                    this->checkBox1->CheckedChanged += gcnew
System::EventHandler(this, &Form1::checkBox1_CheckedChanged);
                    //
                    // groupBox2
                    //
                    this->groupBox2->Controls->Add(this->radioButton1);
                    this->groupBox2->Controls->Add(this->radioButton2);
                    this->groupBox2->Controls->Add(this->radioButton3);
                    this->groupBox2->Location =
System::Drawing::Point(491, 15);
                    this->groupBox2->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->groupBox2->Name = L"groupBox2";
                    this->groupBox2->Padding =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->groupBox2->Size = System::Drawing::Size(467,
295);
                    this->groupBox2->TabIndex = 2;
                    this->groupBox2->TabStop = false;
                    this->groupBox2->Text = L"Cruzamento 2";
                    //
                    // radioButton1
                    //
                    this->radioButton1->AutoCheck = false;
                    this->radioButton1->AutoSize = true;
                    this->radioButton1->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->radioButton1->ForeColor =
System::Drawing::Color::Lime;
                    this->radioButton1->Location =
System::Drawing::Point(9, 98);
                    this->radioButton1->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->radioButton1->Name = L"radioButton1";
                    this->radioButton1->Size = System::Drawing::Size(86,
29);
                    this->radioButton1->TabIndex = 0;
                    this->radioButton1->TabStop = true;
                    this->radioButton1->Text = L"Verde";
                    this->radioButton1->UseVisualStyleBackColor = true;
                    //
                    // radioButton2
                    //
                    this->radioButton2->AutoCheck = false;
                    this->radioButton2->AutoSize = true;
                    this->radioButton2->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->radioButton2->ForeColor =
System::Drawing::Color::Yellow;
                    this->radioButton2->Location =
System::Drawing::Point(8, 62);
                    this->radioButton2->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->radioButton2->Name = L"radioButton2";
```

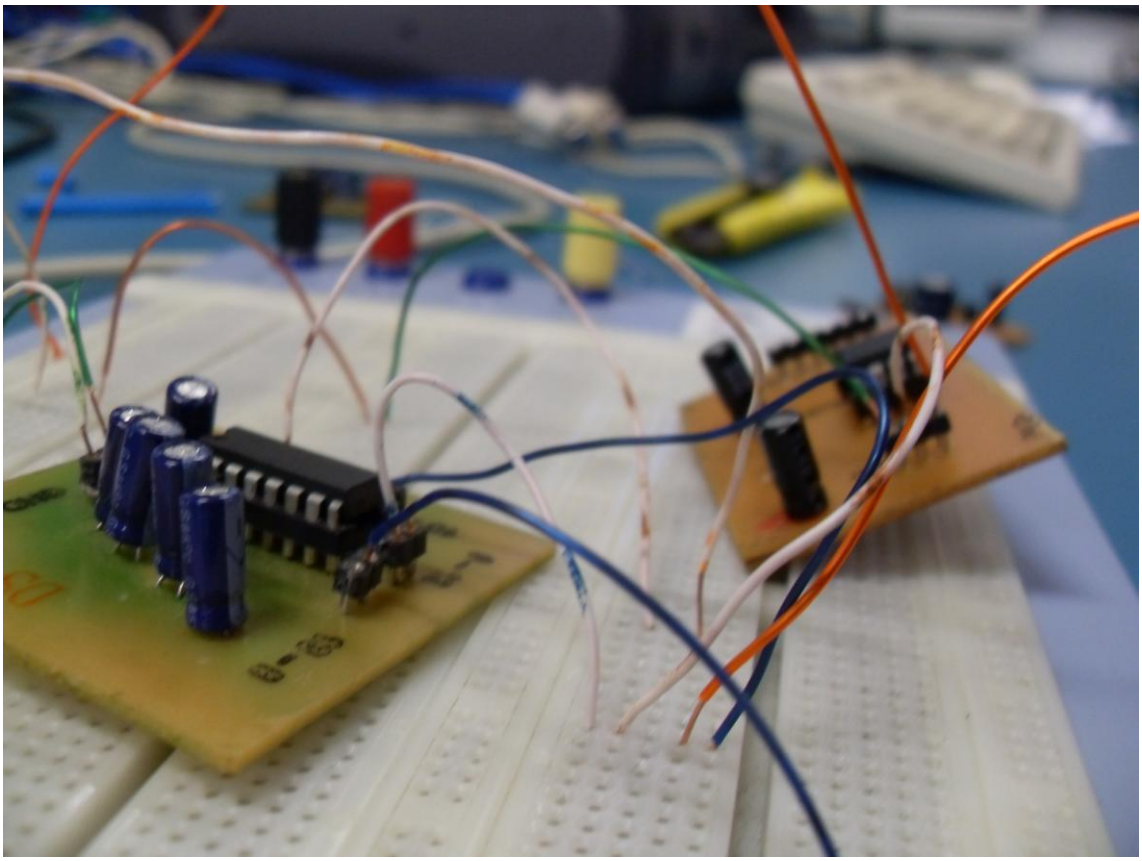
```
                    this->radioButton2->Size = System::Drawing::Size(106,
29);
                    this->radioButton2->TabIndex = 0;
                    this->radioButton2->TabStop = true;
                    this->radioButton2->Text = L"Amarelo";
                    this->radioButton2->UseVisualStyleBackColor = true;
                    //
                    // radioButton3
                    //
                    this->radioButton3->AutoCheck = false;
                    this->radioButton3->AutoSize = true;
                    this->radioButton3->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->radioButton3->ForeColor =
System::Drawing::Color::Red;
                    this->radioButton3->Location =
System::Drawing::Point(9, 25);
                    this->radioButton3->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->radioButton3->Name = L"radioButton3";
                    this->radioButton3->Size = System::Drawing::Size(117,
29);
                    this->radioButton3->TabIndex = 0;
                    this->radioButton3->TabStop = true;
                    this->radioButton3->Text = L"Vermelho";
                    this->radioButton3->UseVisualStyleBackColor = true;
                    this->radioButton3->CheckedChanged += gcnew
System::EventHandler(this, &Form1::radioButton3_CheckedChanged);
                    //
                    // radioButton4
                    //
                    this->radioButton4->AutoCheck = false;
                    this->radioButton4->AutoSize = true;
                    this->radioButton4->Location =
System::Drawing::Point(843, 338);
                    this->radioButton4->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->radioButton4->Name = L"radioButton4";
                    this->radioButton4->Size = System::Drawing::Size(99,
21);
                    this->radioButton4->TabIndex = 3;
                    this->radioButton4->TabStop = true;
                    this->radioButton4->Text = L"Maior em 2";
                    this->radioButton4->UseVisualStyleBackColor = true;
                    //
                    // radioButton5
                    //
                    this->radioButton5->AutoCheck = false;
                    this->radioButton5->AutoSize = true;
                    this->radioButton5->Location =
System::Drawing::Point(721, 338);
                    this->radioButton5->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->radioButton5->Name = L"radioButton5";
                    this->radioButton5->Size = System::Drawing::Size(99,
21);
                    this->radioButton5->TabIndex = 4;
                    this->radioButton5->TabStop = true;
```

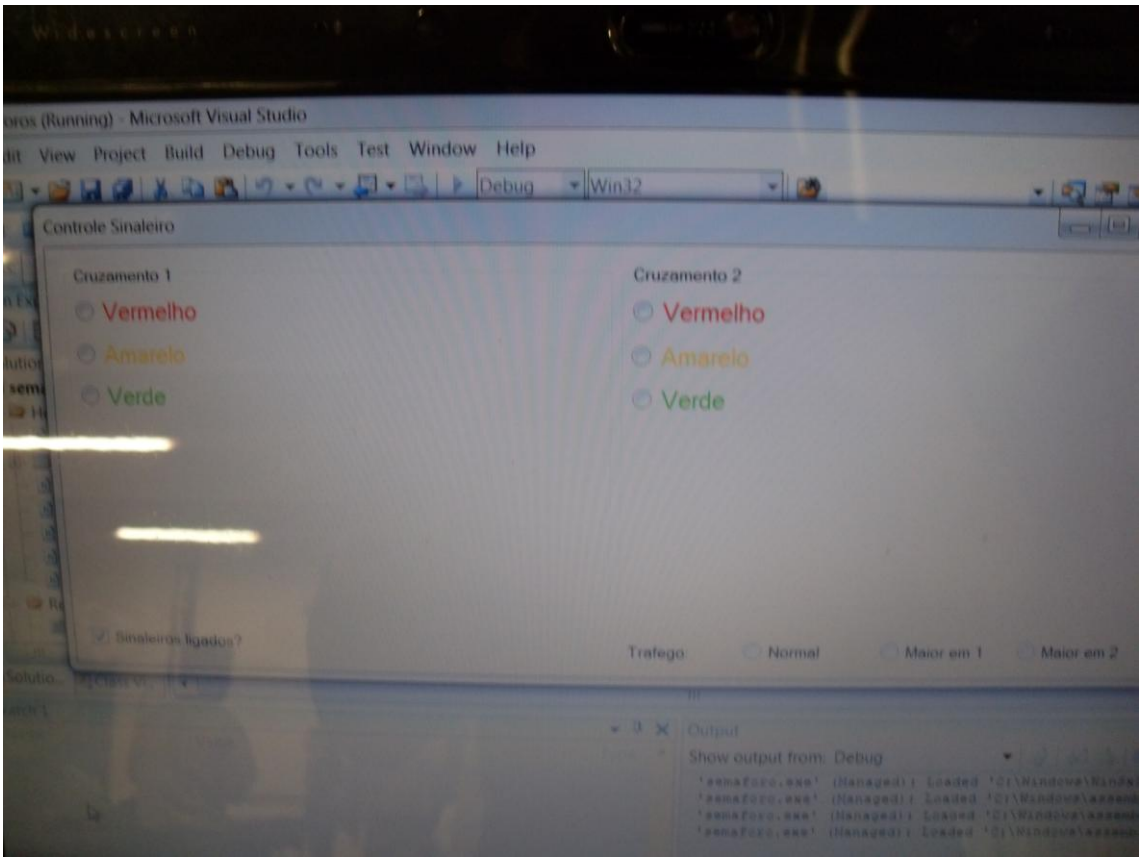
```
                    this->radioButton5->Text = L"Maior em 1";
                    this->radioButton5->UseVisualStyleBackColor = true;
                    //
                    // radioButton6
                    //
                    this->radioButton6->AutoCheck = false;
                    this->radioButton6->AutoSize = true;
                    this->radioButton6->Location =
System::Drawing::Point(600, 338);
                    this->radioButton6->Margin =
System::Windows::Forms::Padding(4, 4, 4, 4);
                    this->radioButton6->Name = L"radioButton6";
                    this->radioButton6->Size = System::Drawing::Size(74,
21);
                    this->radioButton6->TabIndex = 5;
                    this->radioButton6->TabStop = true;
                    this->radioButton6->Text = L"Normal";
                    this->radioButton6->UseVisualStyleBackColor = true;
                    //
                    // label1
                    //
                    this->label1->AutoSize = true;
                    this->label1->Location = System::Drawing::Point(496,
341);
                    this->label1->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
                    this->label1->Name = L"label1";
                    this->label1->Size = System::Drawing::Size(62, 17);
                    this->label1->TabIndex = 6;
                    this->label1->Text = L"Trafego:";
                    //
                    // Form1
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(8,
16);
                    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
                    this->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Center;
                    this->ClientSize = System::Drawing::Size(976, 374);
                    this->Controls->Add(this->label1);
                    this->Controls->Add(this->radioButton6);
                    this->Controls->Add(this->radioButton5);
                    this->Controls->Add(this->radioButton4);
                    this->Controls->Add(this->checkBox1);
                    this->Controls->Add(this->groupBox2);
                    this->Controls->Add(this->groupBox1);
                    this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedDialog;
                    this->Margin = System::Windows::Forms::Padding(4, 4,
4, 4);
                    this->Name = L"Form1";
                    this->Text = L"Controle Sinaleiro";
                    this->Load += gcnew System::EventHandler(this,
&Form1::Form1_Load);
                    this->KeyPress += gcnew
System::Windows::Forms::KeyPressEventHandler(this,
&Form1::Form1_KeyPress);
                    this->KeyUp += gcnew
System::Windows::Forms::KeyEventHandler(this, &Form1::Form1_KeyUp);
```

```
                  this->KeyDown += gcnew
System::Windows::Forms::KeyEventHandler(this, &Form1::Form1_KeyDown);
                  this->groupBox1->ResumeLayout(false);
                  this->groupBox1->PerformLayout();
                  this->groupBox2->ResumeLayout(false);
                  this->groupBox2->PerformLayout();
                  this->ResumeLayout(false);
                  this->PerformLayout();

            }
#pragma endregion
      private: System::Void Form1_KeyDown(System::Object^  sender,
System::Windows::Forms::KeyEventArgs^  e) {


                        }
      private: System::Void Form1_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e) {
                  }
      private: System::Void Form1_KeyUp(System::Object^  sender,
System::Windows::Forms::KeyEventArgs^  e) {

                  }
      private: System::Void Form1_Load(System::Object^  sender,
System::EventArgs^  e) {
                  }
private: System::Void checkBox1_CheckedChanged(System::Object^
sender, System::EventArgs^  e) {

                  if (checkBox1->CheckState == CheckState::Checked)
                  {
                        // If checked, turn on.
                        cruzamento.liga_cruz();
                  }

            }
      private: System::Void
vermelho_aceso_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                  if(cruzamento.semaforo1.vermelha.estado ==
Lampada::ligada){
                        vermelho_aceso->PerformClick();
                  }

            }
private: System::Void radioButton3_CheckedChanged(System::Object^
sender, System::EventArgs^  e) {
                  if(cruzamento.semaforo2.vermelha.estado ==
Lampada::ligada){
                        radioButton3->PerformClick();
                  }
            }
};
      }
```








Microsoft Visual Studio
View Project Build Debug Tools Test Window Help
Debug
Win32
Controle Sinaleiro
Cruzamento 1
Vermelho
Amarelo
Verde
Cruzamento 2
Vermelho
Amarelo
Verde
Sinaleiros ligados?
Trafego
Normal
Maior em 1
Maior em 2
Output
Show output from: Debug