EE226 Lab4
TRAFFIC LIGHT CONTROLLER

MICROPROCESSORS LABORATORY

Ömer Karslıoğlu

Introduction

General Purpose of This Laboratory : Making a traffic light scenario using the final state machine and with SysTick that is the peripheral unit of the microcontroller .

Pin/Port Selection :

OUTPUTS

PE1 , PE2 and PE3 are connected to south leds .
PE1 is connected to red .
PE2 is connected to yellow .
PE3 is connected to green .
(SOUTH PE1-3)

PD1 , PD2 and PD3 are connected to west leds .
PD1 is connected to red .
PD2 is connected to yellow .
PD3 is connected to green .
(WEST PD1-3)

PE4 and PE5 are connected to walking leds .
PE4 is connected to red .
PE5 is connected to green .

Pin/Port Selection :

INPUTS

PA2 , PA3 and PA4 are connected to switches .

Circular reading value is 1 . I used pull up ressistor .

If switches are pressed , the data is read as 0 .

All of them are active low .

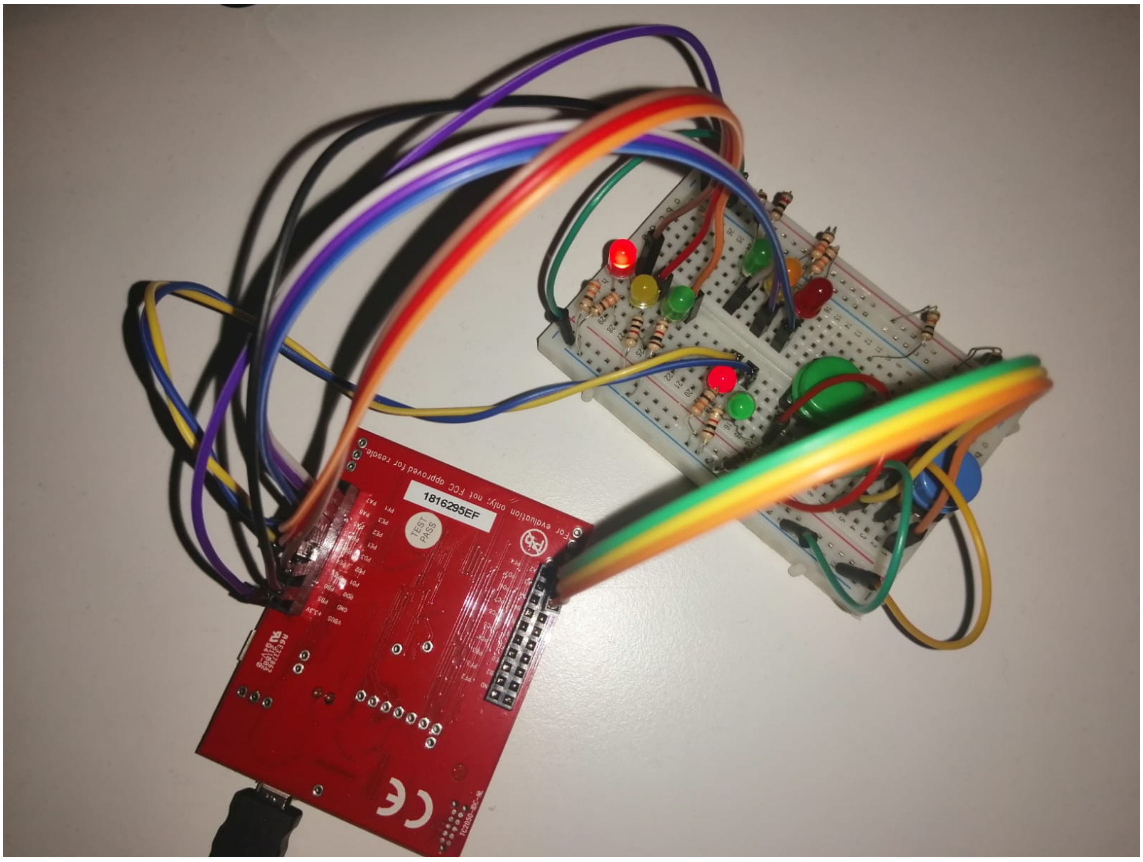

Figure 1 : Traffic Lights Circuit

Designing Final State Machine

Considering all the first diagnostic situations, I determined the outputs as follows.

Note that the format of the leds is :
Walking Leds (GREEN RED)-South Leds (GREEN YELLOW RED) - West Leds (GREEN YELLOW RED)

		Walking/South leds	west leds
Cars Going South	: goS	(01_100_0)	(001_0)
Stop South Cars	: waitS	(01_010_0)	(001_0)
Cars Going West	: goW	(01_001_0)	(100_0)
Stop West Cars	: waitW	(01_001_0)	(010_0)
Walk	: walk	(10_001_0)	(001_0)
WalkFast	: walkFast	(10_001_0)	(001_0)
Lights All Lamps to Red	: allRed	(01_001_0)	(001_0)

Designing Final State Machine

Later, I named all the input states as you can see below.

States according to Inputs

000=No padestrian/No cars

001= No padestrian/South car

010=No pedetrian/West car

011= No pedetrian/South&west car

100=Pedetrian/No cars

101=Pedestrian/South car

110=Pedestrian/West car

111=Pedestrian/South&west cars

Designing Final State Machine

Later, taking the data from the inputs and the previous state, I wrote all the next states step by step into a table in the word file. I thought of all the diagnostic situations while writing the next states. The table is as you can see below.

Inputs : ---------	000	001 (s)	010 (west)	011	100 (walk)	101	110	111
goS	waitS	goS	waitS	waitS	WaitS	waitS	waitS	waitS
waitS	goW	goS	goW	goW	**allRed**	**allRed**	**allRed**	**allRed**
goW	waitW	waitW	goW	waitW	waitW	waitW	waitW	waitW
waitW	**allRed**	goS	goW	goS	**allRed**	**allRed**	**allRed**	goS
walk	walkFast	walkFast	walkFast	walkFast	walk	walkFast	walkFast	walkFast
walkFast	goS	goS	goW	goS	walk	goS	goW	goW
allRed	walk	goS	goW	goW	walk	walk	walk	walk

Figure 2 : FSM Map

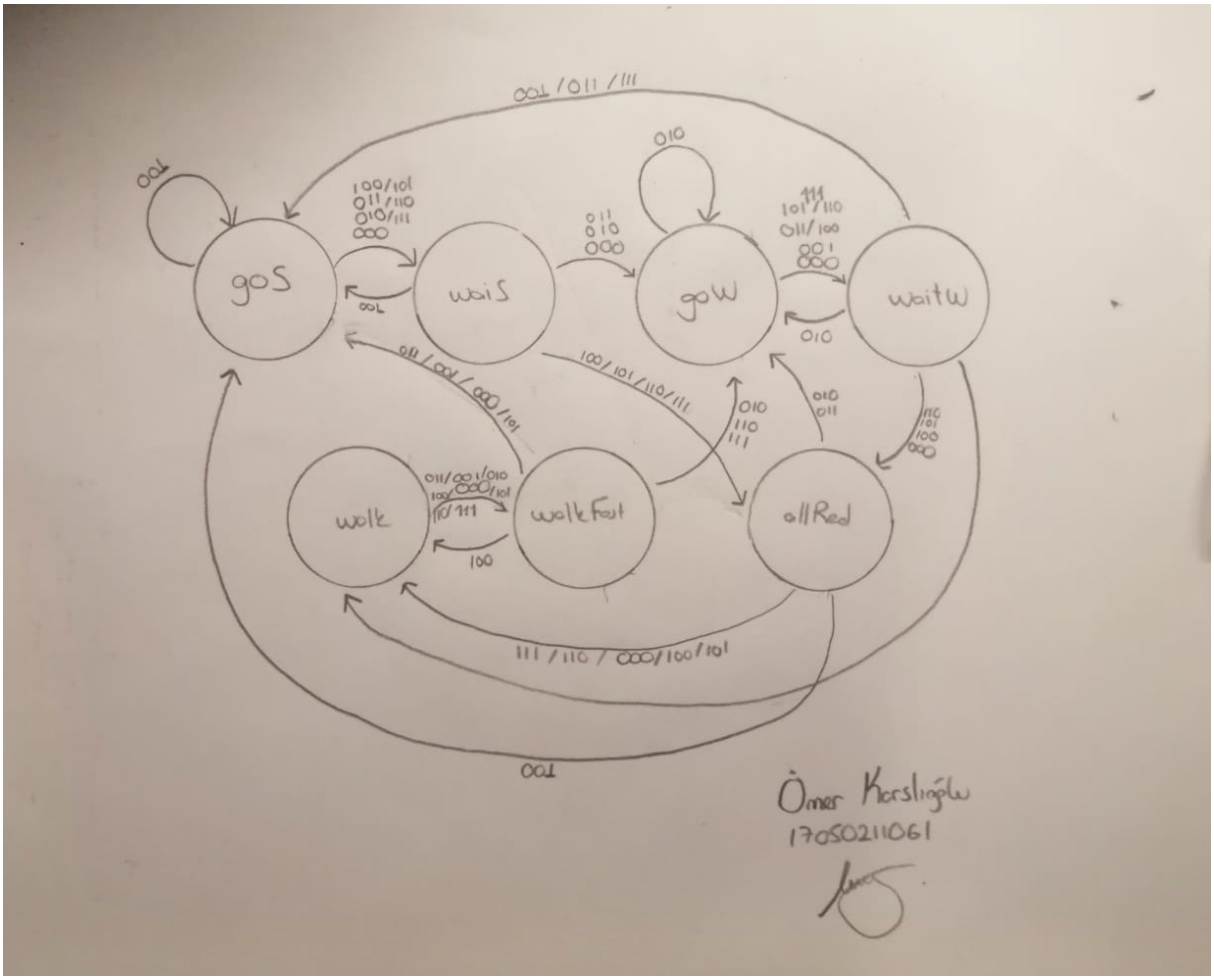

Figure 3 : FSM (Shorthand Format)

Explanations of Code

Earlier in my laboratory reports, I explained in detail what the libraries I use here do, and how to initialize the GPIO.After defining the necessary libraries and defining the GPIO initializing functions, I made my specific address definitions and custom value definitions.

```
#define southLeds          (*((volatile unsigned long *)0x40024338))  // pin specified address (South LEDS)        PE123
#define westLeds           (*((volatile unsigned long *)0x40007338))  // pin specified address (West LEDS)         PD123
#define walkingLeds        (*((volatile unsigned long *)0x400243C0))  // pin specified address (Walking LEDS) PE45

#define inputs             (*((volatile unsigned long *)0x40004370))  // PA2-PA3
```

Figure 4 : Spescific address definition of inputs and outputs

Later, I wrote the SysTick initialize function and related delay functions.

```
void SysTick_Init(void)
{
  NVIC_ST_CTRL_R=0; // disable the st_ctrl register
  NVIC_ST_RELOAD_R =0x00FFFFFF; // maximum reload value
  NVIC_ST_CURRENT_R=0; // clear the current register
  NVIC_ST_CTRL_R=0x00000005; // enable the st_ctrl register
}
```

Figure 5 : SysTick Initializing

When initializing, it is important to disable the control register and enable it later.

CURRENT Register: NVIC_ST_CURRENT_R

Reading it returns the current value of the counter

➢When it transits from 1 to 0, it generates an interrupt if INTEN=1

➢Writing to NVIC_ST_CURRENT_R clears the counter and COUNTFLAG to zero

  ❖Cause the counter to reload on the next timer clock

➢It has random value on reset.

  ❖Always clear it before enabling the timer

(supposing)Clock source = 80 mhz (12.5 ns)
SysTick Interval = 10 ms
(reload+1)*CLK_PERIOD=10ms
reload = (10 ms x 80 MHz)-1
reload = 8x10^6-1 = 799999

```
void SysTick_Wait(unsigned long delay)
{
  NVIC_ST_RELOAD_R=delay;
  NVIC_ST_CURRENT_R=0; // clear the current register
  while((NVIC_ST_CTRL_R & 0x10000)==0){} // waiting until that flag is set
}

//wait for N*1ms
void SysTick_Wait_N_ms(unsigned long N)
{
  unsigned long i;
  for(i=0;i<N;i++){
    SysTick_Wait(80000-1); // 1 ms / 12.5ns = 80000 ==> i obtained 1 ms
  }
}
```

Figure 6 : SysTick Delay Functionality

SysTick_Wait(80000-1) => 1 ms / 12.5ns = 80000 ==> I obtained 1 ms

Final State Machine in Code

```
enum lampColors{
  tlRed        =   0x02,
  tlYellow     =   0x04,
  tlGreen      =   0x08,
  wlRed        =   0x10,
  wlGreen      =   0x20
};

enum states {
  goS          =   0,
  waitS        =   1,
  goW          =   2,
  waitW        =   3,
  walk         =   4,
  walkFast     =   5,
  allRed       =   6
};
```

Figure 7 : Definitions with "enum"

With the "enum" structure, the main states and the led states to be transferred to the ports are defined.

```
struct State
{
  unsigned long walkingTrafficLights;
  unsigned long southTrafficLights;
  unsigned long westTrafficLights;
  unsigned long time;
  unsigned long next[8];
};typedef const struct State SType;
```

Figure 8 : Creating Type

A struct named SType was defined according to the parameters. In this way, I easily transferred the final state machine table we created above to our code.

```
SType FSM[7] =
{
{wlRed  ,tlGreen  ,tlRed    ,3000 ,{waitS ,goS  ,waitS  ,waitS  ,waitS  ,waitS  ,waitS  ,waitS}},
{wlRed  ,tlYellow ,tlRed    ,500  ,{goW ,goS  ,goW  ,goW  ,allRed ,allRed ,allRed ,allRed}},
{wlRed  ,tlRed    ,tlGreen  ,3000 ,{waitW ,waitW  ,goW  ,waitW  ,waitW  ,waitW  ,waitW  ,waitW}},
{wlRed  ,tlRed    ,tlYellow ,500  ,{allRed  ,goS  ,goW  ,goS  ,allRed ,allRed ,allRed ,goS}},
{wlGreen,tlRed    ,tlRed    ,3000 ,{walkFast  ,walkFast ,walkFast ,walkFast ,walk ,walkFast ,walkFast ,walkFast}},
{wlGreen,tlRed    ,tlRed    ,200  ,{goS ,goS  ,goW  ,goS  ,walk ,goS  ,goW  ,goW}}, // walkFast State
{wlRed  ,tlRed    ,tlRed    ,100  ,{walk  ,goS  ,goW  ,goW  ,walk ,walk ,walk ,walk}} // allRed state
};
```

Figure 9 : Creating FSM

I set the green light on times as 3 seconds, the yellow light on as 0.5 seconds, the duration of the allRed state as 0.1 seconds, and the flashing interval of the walkFast state as 0.2 seconds.

Main Part of My Code

```c
int main(void)
{
init_Clock();
init_PortA();
init_PortD();
init_PortE();

SysTick_Init();

state = goS;
walkFastFlagState=0;
```

The previous state when the master code first started is goS.

```c
while(1)
{
  southLeds    = FSM[state].southTrafficLights;
  westLeds     = FSM[state].westTrafficLights;
  if(state==walkFast){
    for(i=0;i<=5;i++){
      walkingLeds = (walkFastFlagState) ? 0x00 : 0x10 ;
      walkFastFlagState = !(walkFastFlagState);
      SysTick_Wait_N_ms(FSM[state].time);
    }
  }
  else{
    walkingLeds = FSM[state].walkingTrafficLights;
    SysTick_Wait_N_ms(FSM[state].time);
  }
  //........ inputu oku
  sensorvalue =(inputs>>2) ^ 0x07 ;
  state=FSM[state].next[sensorvalue];
}

return 0;
}
```

Here is the part that runs the final state machine.

southLeds, westLeds and walkingLeds address
data according to the current state (from the final state machine kept as SType) transfer is performed.

The system continues to work by adjusting the delay times thanks to the SysTick_Wait_N_ms () function.

The main reason the code is separated as if-else is to make the walking led toggle red in the walkFast state.In the walkFast state, it does a total of five red led toggles.

Looking at the state and the data from the sensors, the next state occurs after reading the data from the inputs.

ALL CODES

```c
#include <stdlib.h>
#include <stdint.h>

#include <tm4c123gh6pm.h>

/*
OUTPUTS
-------
Definition : PE1 , PE2 and PE3 are connected to south leds .
PE1 is connected to red .
PE2 is connected to yellow .
PE3 is connected to green .

PD1 , PD2 and PD3 are connected to west leds .
PD1 is connected to red .
PD2 is connected to yellow .
PD3 is connected to green .

PE4 and PE5 are connected to walking leds .
PE4 is connected to red .
PE5 is connected to green .

INPUTS
------
PA2 , PA3 and PA4 are connected to switches .
Circular reading value is 1 . I used pull up ressistor .
If switches are pressed , the data is read as 0 .
All of them are active low .
*/
#define southLeds             (*((volatile unsigned long *)0x40024338))  // pin specified address (South LEDS)         PE123
#define westLeds              (*((volatile unsigned long *)0x40007338))  // pin specified address (West LEDS)          PD123
#define walkingLeds           (*((volatile unsigned long *)0x400243C0))  // pin specified address (Walking LEDS) PE45

#define inputs                (*((volatile unsigned long *)0x40004370))  // PA2-PA3


void init_PortA(void);
void init_PortD(void);
void init_PortE(void);

void init_Clock(void);

void SysTick_Init(void);
void SysTick_Wait(unsigned long delay);
void SysTick_Wait_N_ms(unsigned long N);


enum lampColors{
  tlRed      =   0x02,
  tlYellow   =   0x04,
  tlGreen    =   0x08,
  wlRed      =   0x10,
  wlGreen    =   0x20
};

enum states {
  goS        =   0,
  waitS      =   1,
  goW        =   2,
  waitW      =   3,
  walk       =   4,
  walkFast   =   5,
  allRed     =   6
};


struct State
{
  unsigned long walkingTrafficLights;
  unsigned long southTrafficLights;
  unsigned long westTrafficLights;
  unsigned long time;
  unsigned long next[8];
};typedef const struct State SType;


SType FSM[7] =
{
{wlRed  ,tlGreen  ,tlRed    ,3000 ,{waitS ,goS  ,waitS  ,waitS  ,waitS  ,waitS  ,waitS  ,waitS}},
{wlRed  ,tlYellow ,tlRed    ,500  ,{goW ,goS  ,goW  ,goW  ,allRed ,allRed ,allRed ,allRed}},
{wlRed  ,tlRed    ,tlGreen  ,3000 ,{waitW ,waitW  ,goW  ,waitW  ,waitW  ,waitW  ,waitW  ,waitW}},
{wlRed  ,tlRed    ,tlYellow ,500  ,{allRed  ,goS  ,goW  ,goS  ,allRed ,allRed ,allRed ,goS}},
{wlGreen,tlRed    ,tlRed    ,3000 ,{walkFast  ,walkFast ,walkFast ,walkFast ,walk ,walkFast ,walkFast ,walkFast}},
{wlGreen,tlRed    ,tlRed    ,200  ,{goS ,goS  ,goW  ,goS  ,walk ,goS  ,goW  ,goW}}, // walkFast State
{wlRed  ,tlRed    ,tlRed    ,100  ,{walk  ,goS  ,goW  ,goW  ,walk ,walk ,walk ,walk}} // allRed state
};
```

```
unsigned long state ;
unsigned int sensorvalue ;
int walkFastFlagState;
int i;


int main(void)
{
init_Clock();
init_PortA();
init_PortD();
init_PortE();

SysTick_Init();

state = goS;
walkFastFlagState=0;

while(1)
{
  southLeds    = FSM[state].southTrafficLights;
  westLeds     = FSM[state].westTrafficLights;
  if(state==walkFast){
    for(i=0;i<=5;i++){
      walkingLeds = (walkFastFlagState) ? 0x00 : 0x10 ; // green - red (toggle)
      walkFastFlagState = !(walkFastFlagState);
      SysTick_Wait_N_ms(FSM[state].time);
    }
  }
```

```
    }
  }
  else{
    walkingLeds = FSM[state].walkingTrafficLights;
    SysTick_Wait_N_ms(FSM[state].time);
  }
  //........ inputu oku
  sensorvalue =(inputs>>2) ^ 0x07 ;
  state=FSM[state].next[sensorvalue];
}

return 0;
}
/*
I : input
O : output
*/
void init_PortA(void)
{
  GPIO_PORTA_LOCK_R  = 0x4C4F434B;       // unlock GPIO Port A
  GPIO_PORTA_CR_R    = 0x3F;             // allow changes to PA4-PA2

  GPIO_PORTA_DIR_R   = 0x00;             //00000 <=> I IIII
  GPIO_PORTA_DEN_R   = 0x3F;             //01110  PA2 , PA3 and PA4 are ACTIVETED
}
void init_PortD(void)
{
  GPIO_PORTD_LOCK_R  = 0x4C4F434B;       // unlock GPIO Port D
  GPIO_PORTD_CR_R    = 0x0E;             // allow changes to PD3-PD1
  GPIO_PORTD_LOCK_R  = 0x4C4F434B;       // unlock GPIO Port D
  GPIO_PORTD_CR_R    = 0x0E;             // allow changes to PD3-PD1

  GPIO_PORTD_DIR_R   = 0x0E;             //01110 <=> I OOOI
  GPIO_PORTD_DEN_R   = 0x0E;             //01110  PD1 , PD2 , PD3 ACTIVE

  westLeds =0x00;                            //specified address
}
void init_PortE(void)
{
  GPIO_PORTE_LOCK_R  = 0x4C4F434B;       // unlock GPIO Port E
  GPIO_PORTE_CR_R    = 0x3F;             // allow changes to PE5-PE0

  GPIO_PORTE_DIR_R   = 0x3F;             //1 1111 <=> I OOOI
  GPIO_PORTE_DEN_R   = 0x3F;             //0011111  PE1 , PE2 , PE3 , PE4 , PE5 ACTIVE

  southLeds=0x00;                            //specified address
  walkingLeds=0x00;
}
void init_Clock(void)
{
  SYSCTL_RCGC2_R |= 0x19;                // to initialize clock to Port A , Port D and Port E
  while(!( SYSCTL_PRGPIO_R& 0x19)){ }    //waiting to enable clk
}

void SysTick_Init(void)
{
  NVIC_ST_CTRL_R=0; // disable the st_ctrl register
  NVIC_ST_RELOAD_R =0x00FFFFFF; // maximum reload value
```

```c
{
  NVIC_ST_CTRL_R=0; // disable the st_ctrl register
  NVIC_ST_RELOAD_R =0x00FFFFFF; // maximum reload value
  NVIC_ST_CURRENT_R=0; // clear the current register
  NVIC_ST_CTRL_R=0x00000005; // enable the st_ctrl register
}

//(supposing)Clock source = 80 mhz (12.5 ns)
//SysTick Interval = 10 ms
//(reload+1)*CLK_PERIOD=10ms
//reload = (10 ms x 80 MHz)-1
//reload = 8x10^6-1 = 799999

void SysTick_Wait(unsigned long delay)
{
  NVIC_ST_RELOAD_R=delay;
  NVIC_ST_CURRENT_R=0; // clear the current register
  while((NVIC_ST_CTRL_R & 0x10000)==0){} // waiting until that flag is set
}

//wait for N*1ms
void SysTick_Wait_N_ms(unsigned long N)
{
  unsigned long i;
  for(i=0;i<N;i++){
    SysTick_Wait(80000-1); // 1 ms / 12.5ns = 80000 ==> i obtained 1 ms
  }
}
// Omer Karslioglu - 17050211061
```

Thanks for reading ...

I added to my video link at page 2 of the my lab4 report.