IP540 Full Stack Embedded Software Engineer Intern Training Program

www.ctione.org

Silicon Valley Education And Innovation Association LLC

Version 1.0

Fall 2018

Coordinator: Harry Li, Ph.D. Director
(650) 400-1116, Email: harry.li@ctione.com
CTI One Corporation
3679 Enochs Street
Santa Clara, CA 95051

Contributors

2017-10	Establish this document	Harry Li, Ph.D.
2018-10	Add MOTDRV Chapter	Zhixuan Zhou
2018-10	Add Ethernet and IP networking Chapter	Jerry Lee

Table of Content

I. Full Stack Embedded Software Engineer Training 6 Project Guidelines

Project	Description	Assessment Requirement
1. Getting Started with GPIO Interface Design and Implementation	1. Build the prototype board to realize the function of power regulator circuit, and GPP input and output testing circuit; 2. Install MCU xpresso (the newer version) or Xpresso (you can still use it). Then Import the testing program, GPIO testing program, be sure to import 1769 patch program so the GPIO testing program can run properly; 3. Compile and build the program, then test your prototype design. Once it worked, make demo.	See Rubrics
2. ExINT and INT Timer Design and Implementation	Design a prototype board using LPC1769 CPU module to build INT timer to generate 20-200 mSec waveform output. Your prototype board should have the following function: 1. Use IDE console input to define interrupt timing requirements. 2. Generate timing waveform based on the INT Timer and use logical analyzer to display the waveform. 3. Write a report and make a demo.	
3. Two Dimensional Graphics Process Engine Design and Implementation	Design and prototype LPC1769 micro-processor system board, and enable one SPI LCD display with LPC1769 CPU. Use draw line test code to run draw-a-line testing program from you LPC1769-LCD node. Implement 2D screen saver functions 1 and 2. Function 1 is for rotating square patterns and Function 2 is for generating trees.	See Rubrics

4. Industrial IoT and Wireless Communications		
5. Network and IP Communications		
6. Sensors and MySQL database Interface		

II. Full Stack Embedded Software Engineer
Intern Training Program Schedule (12 Weeks)

The schedule is subject to change with fair notice in class.

Table 2.1 Schedule

Week	Topics, Readings, Assignments, Deadlines
1	Organizational Meeting and Introduction, Overview of a RISC Microprocessor System with CPU datasheet.
2	Review of the RISC CPU architecture: CPU core, peripheral controllers, internal buses, and memory controllers as well as graphics engine. Design a microprocessor system with CPU module and with RS232 debugging capability.
3	Continue to review RISC CPU and peripheral controllers, in particular to UART and SPI controller for design and building FLASH memory interface. I/O Interface Design, from UART, RS232 to RS485, SPI and IIC interface.
4	Memory Map, Power-up Address, ROM memory unit and its 8-bit, 16-bit, 32-bit banks design implementation. Description and design of a bidirectional system bus and I/O. System memories: SRAM and SRAM bus interface.
5	SPI Interface and SPI FLASH memory interface design, implementation of data logger based on SPI FLASH memory design.
6	External controller and implementation of ExINT techniques. Interrupt techniques and Timer design as well as applications based on interrupt timer design. Interrupt controller design and interface design. Advanced MCU design: GE (graphics engine) design, 2D graphics vector graphics processing for ARM Cortex Core. LCD Display adapter design. Implementation of LCD display with 2D GE vector graphics.
7	CPU Architecture Theory, Von Neumann Architecture. Intel CPU interrupt techniques, interrupt vector table, interrupt service routine (ISR) implementations.
8	Midterm
9	ADC interface design and FFT based data validation.
10	Advanced MCU design: GE (graphics engine) design, 3D graphics vector graphics processing, world to viewer transformations, and linear decoration algorithm for ARM Cortex Core. Implementation of 3D perspective projection and linear decoration algorithm.
11	Inter processor communication design with Arduino CPU via serial link.

Week	Topics, Readings, Assignments, Deadlines
12	GE (graphics engine) design, 3D graphics vector graphics processing, texture mapping and animation implementation.
13	ExTNT technique integration with LCD display and 3D GE acceleration engine.
14	Inter-processor communications with Arduino CPU and PWM based laser imaging radar.
15	Advanced inter-processor communications, PWM based laser imaging radar with ARM CPU USB based camera image data acquisition.
16	Comprehensive final exam.

III. Getting Started with GPIO Interface Design and Implementation

3.1 Design requirements

1. Build the prototype board to realize the function of power regulator circuit, and GPP input and output testing circuit;

2. Install MCU xpresso (the newer version) or Xpresso (you can still use it). Then Import the testing program, GPIO testing program, be sure to import 1769 patch program so the GPIO testing program can run properly;

3. Compile and build the program, then test your prototype design. Once it worked, make demo.

3.2 Assessment

1. General Guidelines

Use IEEE paper template to generate your report.

(1) One line title, factual and right-to-the-point, avoid marketing terms.

(2) Abstract (50-100 words) with design objectives, technical challenges,

methodology, hardware and software resources description, deliverables,

and the implementation result.

(3) Strickly follow the IEEE paper style, no modification of spacing, fonts,

section enumeration etc. be sure to provide Appendix section with source code and schematics.

2. System and Hardware Level Design

(1) provide system block diagram to capture the entire desing/testing/prototyping setup, for example, laptop computer with Ethernet/RS232 link to microprocessor system.

(2) Block diagram for the micropocessor system with detailed pin connectivity information, labels of each individual block of the system.

(3) Schematics of each basic building blocks and/or subsystems, and/or entire system.

(4) Photos of the actual implementation of the entire system and/or subsystems.

3. Software Design

(1) Description of the softweare development environment and its set up procedure, such as Eclips based IDE (integrated development environment) set

up.

(2) Algorithm description in a well-organized, step-by-step fashion, for example steps from 1 through 5.

(3) Flow chart(s) to give further details of the algorithm, if needed, multiple flow charts can be utilized.

(4) Pseudo code to match up the flow charts. Due to the nature of the hardware and software co-design, algorithmic type Pseudo code is usually too abstract, details down to the level of registers and bits patterns of registers are needed.

(5) Source code (segment of code) to support the Pseudo Code.

4. Testing and Verification

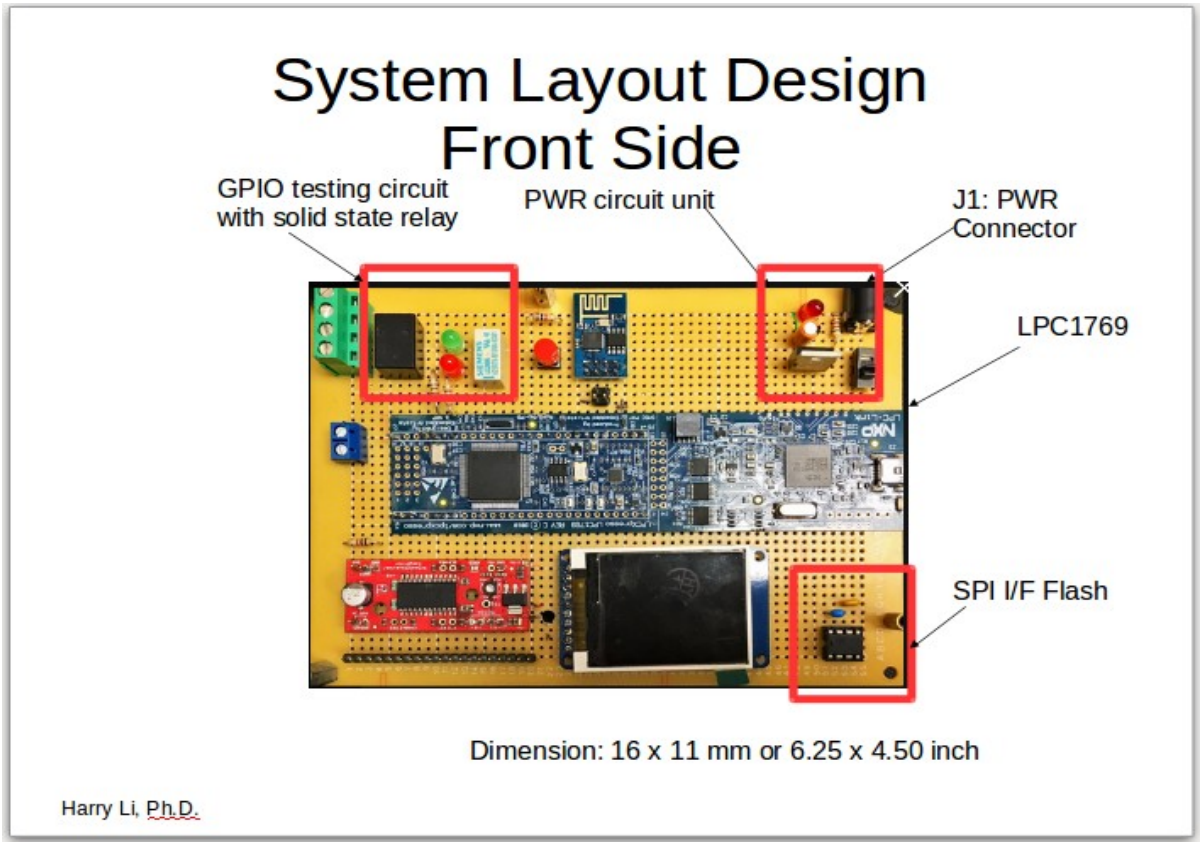
Report will have a Testing and Verification section, which will cover

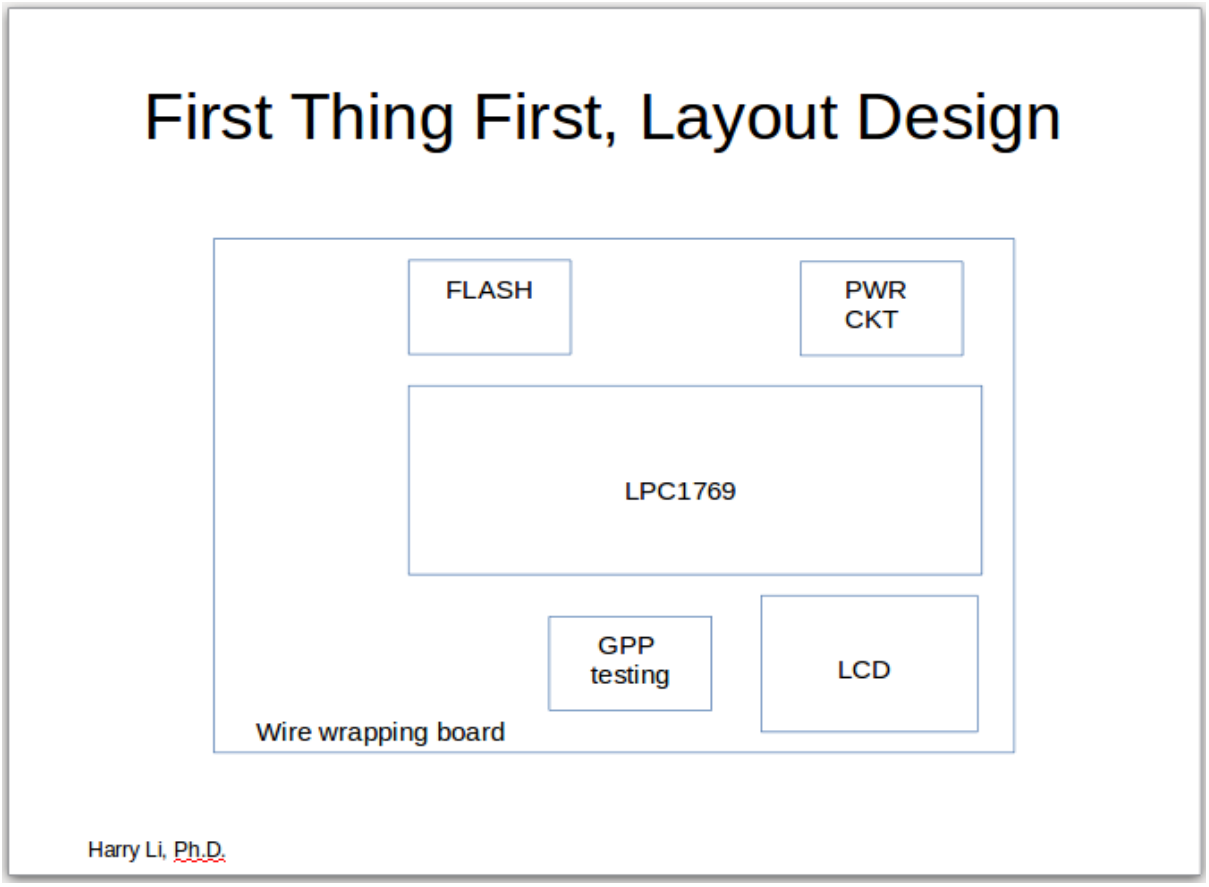
(1) Hardware testing: photos of the waveforms from oscilloscope and/or logic analyzer. Provide data from the system testing result, and/or SPICE simulation capture if needed.

(2) Software testing: Screen capture of the execution result, data from program execution.

5. Reference section with detailed technical reference and datasheet, etc.

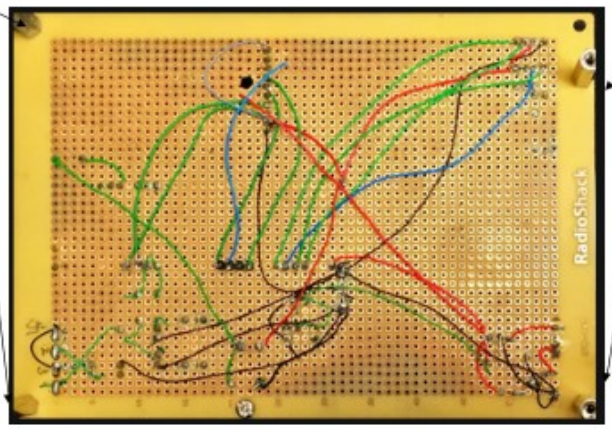
3.3 Design Notes





System Layout Design Back Side

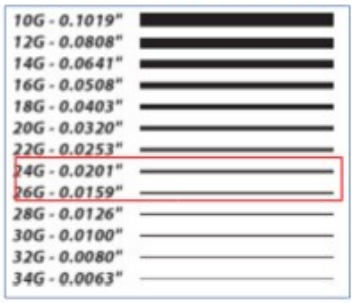
Standoffs



Standoffs

Wire soldered or wire wrapped

Wire Wrapping: 18g to 24G



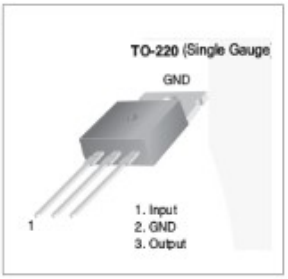
http://www.softflexcompany.com/WSWrapper.jsp?mypage=Tips_Wire_Temper.html

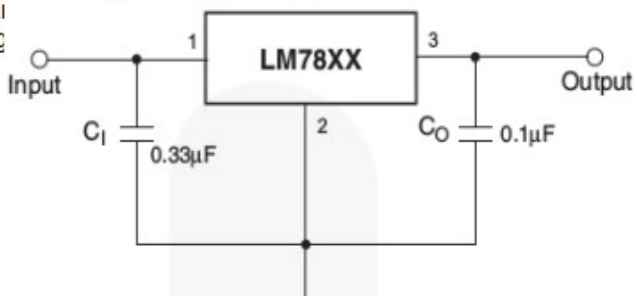
PWR Regulator LM7805

FAIRCHILD SEMICONDUCTOR

LM78XX / LM78XXA
3-Terminal 1 A Positive Voltage Regulator

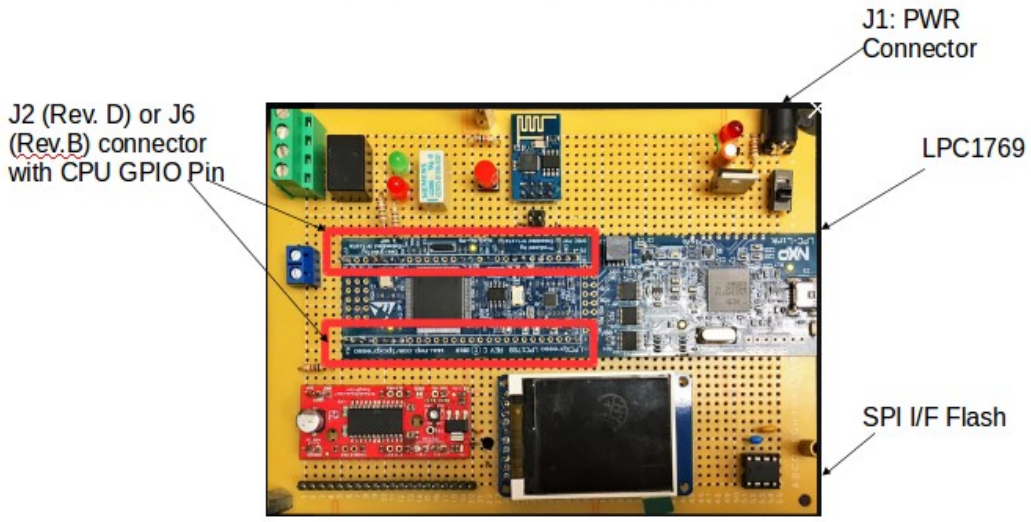
The LM78XX series of three-terminal positive regulators is available in the TO-220 package and with several fixed output voltages, making them useful in a wide range of applications. Each type employs internal current limiting, thermal shut-down, and safe operating area protection. If adequate heat sinking is provided, they can deliver over 1 A output current. Although designed primarily as fixed-voltage regulators, these devices can nal components for adjustable voltag





Note: 1. Voltage drop 1.5 volt;
2. current output 1 A.

J2/(or 6 for rev. B) Connector with CPU GPIO Pins



Dimension: 16 x 11 mm or 6.25 x 4.50 inch

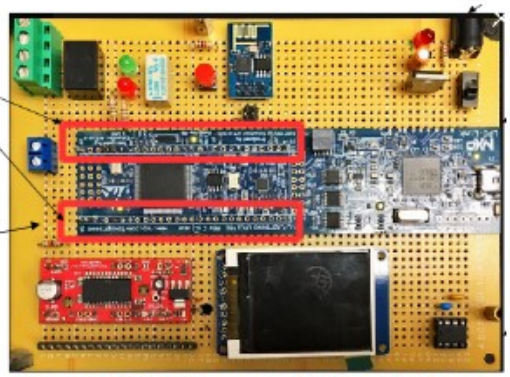
Harry Li, Ph.D.

J2 (Rev. D) or J6 (Rev. B) Connector with CPU GPIO Pins

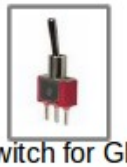
Table 1. J2 Pin Assignment

P1.31 AD0.5	P1.31	J2-20
P0.2	P0.2	J2-21
P0.3	P0.3	J2-22
P0.21	P0.21	J2-23
P0.22	P0.22-RED_LED	J2-24
P0.27	P0.27-I2C_SDA	J2-25
P0.28	P0.28-I2C_SCL	J2-26
P2.13	P2.13	J2-27

Reference: SCH design Rev D.



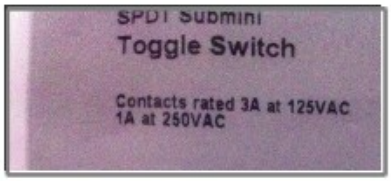
Board: 16 x 11 mm or 6.25 x 4.50 inch



SPDT switch for GPIO input testing

Table 2. J2 Connectivity

CPU	J2	Description
P0.2	J2-21	GPIO output
P0.3	J2-22	GPIO input

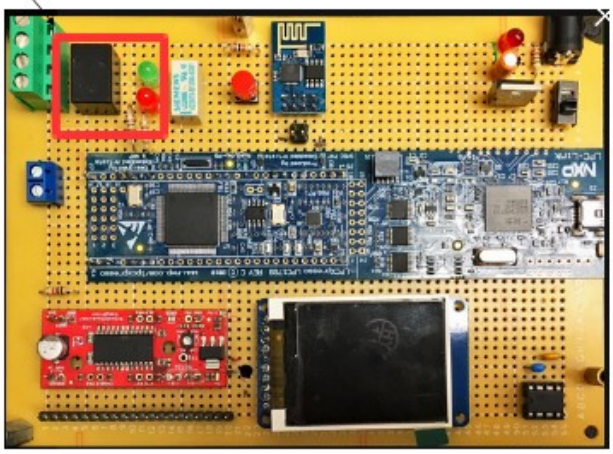


A Single Pole Double Throw (SPDT) switch is a switch that only has a single input and can connect to and switch between 2 outputs.

Harry Li, Ph.D.

GPIO (GPP) Output with SSR

SSR: Solid State Relay

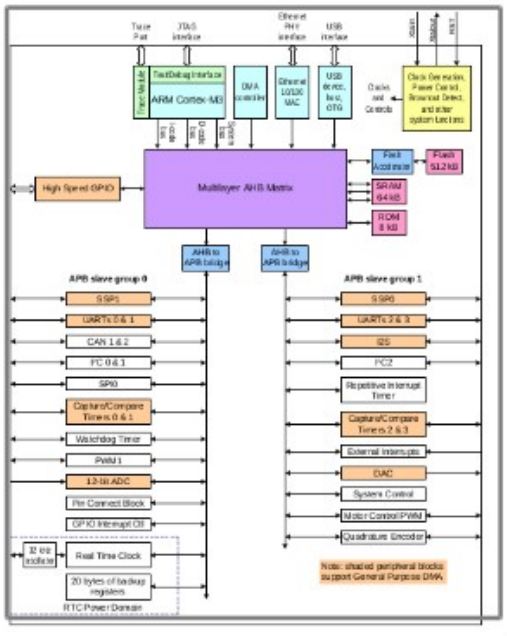


SSR: Solid State Relay

GPIO (GPP) Controller and SPRs

1. Definition of SPRs: Special purpose registers are those to perform init and config functions.
2. 32 bit each with unique address.
3. In IDE (software, e.g., eXpresso in this class), *.h file with something looks like the following
#define SPR 0x2000_0000
map the CPU architecture to the arm gcc compiler

```
LPC_GPIO0->FIODIR

LPC_GPIO0->FIOSET

LPC_GPIO0->FIOCLR
```



CPU

Memory Map

GPIO (GPP) SPRs

GPIO SPRs

Reference: Chapter 9: LPC176x/5x General Purpose Input/Output (GPIO) Rev. 3.1 — 2 April 2014 User manual

```
LPC_GPIO0->FIODIR

LPC_GPIO0->FIOSET

LPC_GPIO0->FIOCLR
```

```
#include "../../../tools/redlib/include/stdint.h"
#include "../../../tools/redlib/include/stdio.h"

#ifdef __USE_CMSIS     //board init & config stuff
 /* CMSIS: the Cortex Microcontroller Software Interface Standard
                        */
#include "LPC17xx.h"
```

Background:

From CPU datasheet, GPIOs are configured using the following registers:
1. Power: always enabled.
2. Pins: See Section 8.3 for GPIO pins and their modes.
3. Wake-up: GPIO ports 0 and 2 can be used for wake-up if needed, see (Section 4.8.8).
4. Interrupts: Enable GPIO interrupts in IO0/2IntEnR (Table 115) or IO0/2IntEnF (Table 117). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

9.4 Pin description P0[30:0] [1] ; Type: Input/Output; Description: General purpose in/output.

From SCH pdf doc make connection from this GPP pin to physical connector pin out, e.g., J2-21, J2-22 etc.

Harry Li, Ph.D

GPIO SPRs Description

```
LPC_GPIO0->FIODIR

LPC_GPIO0->FIOSET

LPC_GPIO0->FIOCLR
```

Table 102. GPIO register map, from CPU data sheet pp133

FIODIR Fast GPIO Port Direction control register, controls the direction of each port pin

FIOSET Fast Port Output Set register using FIOMASK. This register
R/W
controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.

FIOCLR Fast Port Output Clear register using FIOMASK. This register WO controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK can be altered.

FIOxDIR Description

```
LPC_GPIO0->FIODIR

LPC_GPIO0->FIOSET

LPC_GPIO0->FIOCLR
```

Reference: CPU Datasheet, 9.5.1 GPIO port Direction register FIOxDIR (FIO0DIR to FIO4DIR- 0x2009 C000 to 0x2009 C080)

Table 104. Fast GPIO port Direction register FIO0DIR to FIO4DIR - addresses 0x2009 C000 to 0x2009 C080) bit description

FIO0DIR
(Compiler: LPC_GPIO0->FIODIR)

Bit	Symbol	Value	Description
31:0	FIO0DIR		Fast GPIO Direction PORTx control bits. Bit 0 in FIOxDIR controls pin Px.0, bit 31 in FIOxDIR controls pin Px.31.
	FIO1DIR		
	FIO2DIR	0	Controlled pin is input.
	FIO3DIR		
	FIO4DIR	1	Controlled pin is output.

Example: FIO0DIR Init & Config,

Set P0.2 output, "1"
P0.3 input, "0"

0x4


```
LPC_GPIO0->FIODIR = 0x4;
```

Harry Li, Ph.D.

*.h Mapping the CPU Architecture

```
LPC_GPIO0->FIODIR

LPC_GPIO0->FIOSET

LPC_GPIO0->FIOCLR
```

Example: LPC1769.h
Maps compiler to architecture

```
/*--General Purpose Input/Output (GPIO) --*/
typedef struct
{
  union {
    __IO uint32_t FIODIR;
    struct {
      __IO uint16_t FIODIRL;
      __IO uint16_t FIODIRH;
    };
    struct {
      __IO uint8_t  FIODIR0;
      __IO uint8_t  FIODIR1;
      __IO uint8_t  FIODIR2;
      __IO uint8_t  FIODIR3;
    };
  };
//…
//…
} LPC_GPIO_TypeDef;
```

Harry Li, Ph.D.

*.h Mapping the Memory Map

```
LPC_GPIO0->FIODIR

LPC_GPIO0->FIOSET

LPC_GPIO0->FIOCLR
```

Example: LPC1769.h
Maps compiler to memory map

1

Peripheral Memory Map

```
/*****************************************************/
/*      Peripheral memory map                        */
/*****************************************************/
/* Base addresses                                    */
#define LPC_FLASH_BASE        (0x00000000UL)
#define LPC_RAM_BASE          (0x10000000UL)
#define LPC_GPIO_BASE         (0x2009C000UL)
#define LPC_APB0_BASE         (0x40000000UL)
#define LPC_APB1_BASE         (0x40080000UL)
#define LPC_AHB_BASE          (0x50000000UL)
#define LPC_CM3_BASE          (0xE0000000UL)
```

2

```
/* GPIOs */
#define LPC_GPIO0_BASE        (LPC_GPIO_BASE + 0x00000)
#define LPC_GPIO1_BASE        (LPC_GPIO_BASE + 0x00020)
#define LPC_GPIO2_BASE        (LPC_GPIO_BASE + 0x00040)
#define LPC_GPIO3_BASE        (LPC_GPIO_BASE + 0x00060)
#define LPC_GPIO4_BASE        (LPC_GPIO_BASE + 0x00080)
```

3

```
/************ Peripheral declaration **************/
#define LPC_SC                ((LPC_SC_TypeDef       *) LPC_SC_BASE     )
#define LPC_GPIO0             ((LPC_GPIO_TypeDef     *) LPC_GPIO0_BASE  )
#define LPC_GPIO1             ((LPC_GPIO_TypeDef     *) LPC_GPIO1_BASE  )
#define LPC_GPIO2             ((LPC_GPIO_TypeDef     *) LPC_GPIO2_BASE  )
#define LPC_GPIO3             ((LPC_GPIO_TypeDef     *) LPC_GPIO3_BASE  )
#define LPC_GPIO4             ((LPC_GPIO_TypeDef     *) LPC_GPIO4_BASE  )
```

GPIO Init and Config

```
int main(void)
{
    // Force the counter to be
placed into memory
    volatile static int i = 0 ;
    //Set pin 0.2 as output
GPIOinitOut(0,2); //port number,
pin number
//Set pin 0.3 as output
GPIOinitOut(0,3);

setGPIO(0, 3);
clearGPIO(0, 2); //port number,
pin number
```

```
void GPIOinitOut(uint8_t portNum,
uint32_t pinNum)
{
if (portNum == 0)
{
LPC_GPIO0->FIODIR |= (1 <<
pinNum);
}
else if (portNum == 1)
{
LPC_GPIO1->FIODIR |= (1 <<
pinNum);
```

```
void clearGPIO(uint8_t portNum, uint32_t
pinNum)
{
if (portNum == 0)
{
LPC_GPIO0->FIOCLR = (1 << pinNum);
printf("Pin 0.%d has been cleared.\n",
pinNum);
}
```

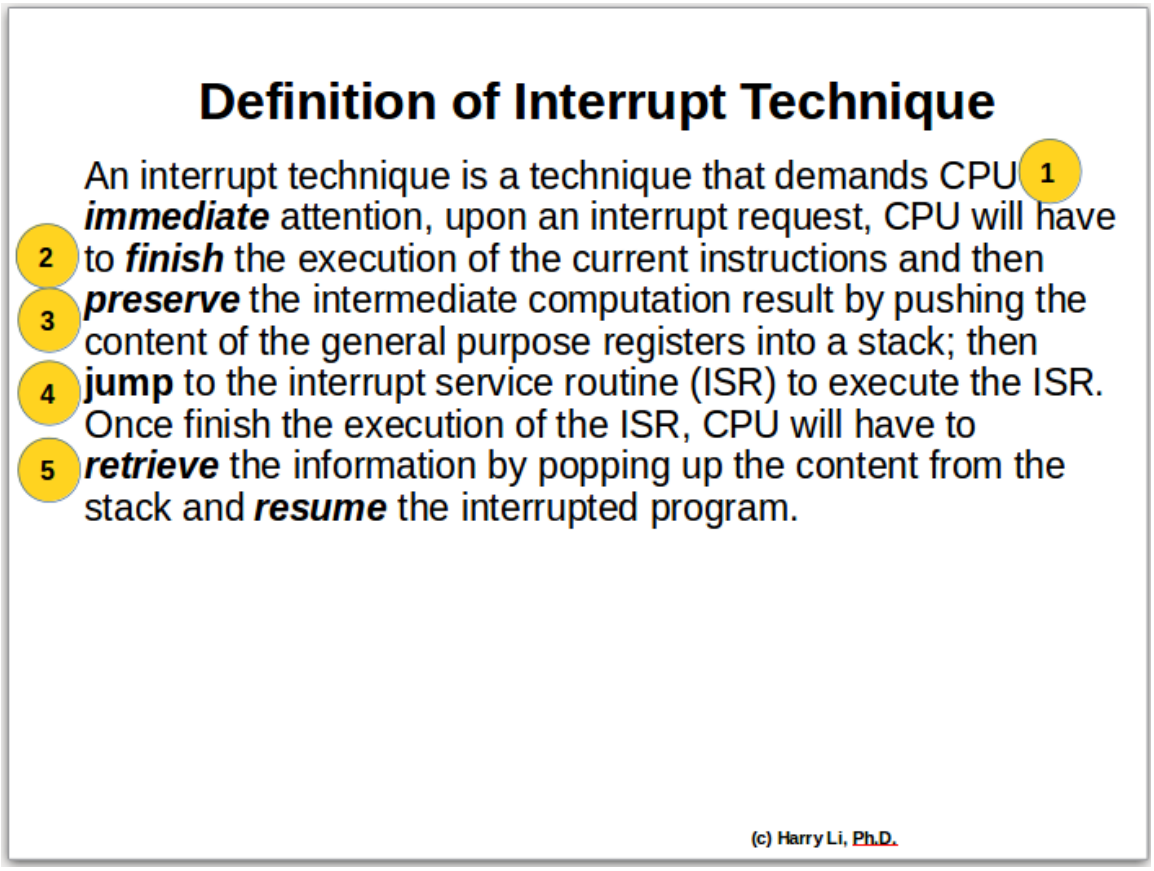
```
void setGPIO(uint8_t portNum,
uint32_t pinNum)
{
if (portNum == 0)
{
LPC_GPIO0->FIOSET = (1 <<
pinNum);   //1 as output
printf("Pin 0.%d has been
set.\n",pinNum);
}
```

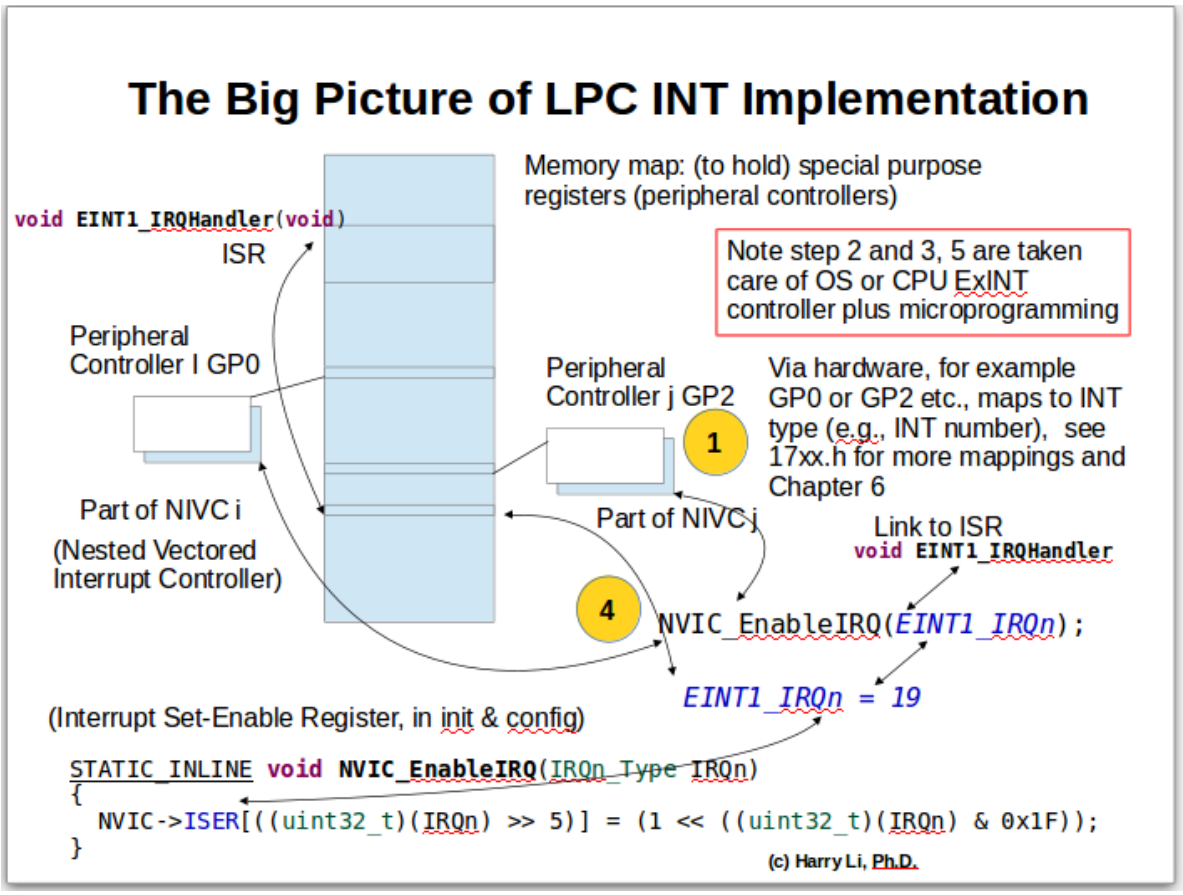
IV. ExINT and INT Timer Design and Implementation

4.1 Design requirements

4.2 Assessment

4.3 Design Notes

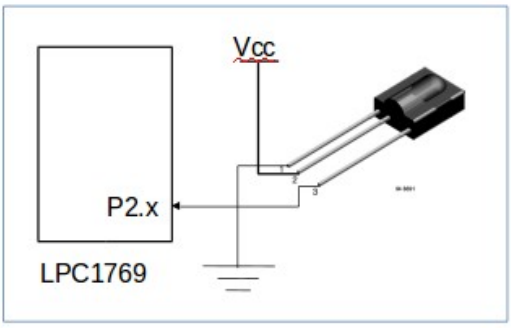


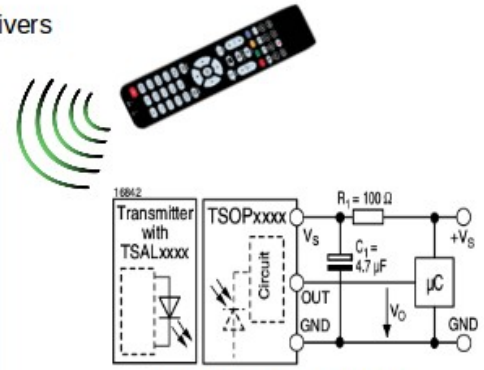


IrDA IC Triggers the ExINT

IR Receiver Modules for Remote Control Systems

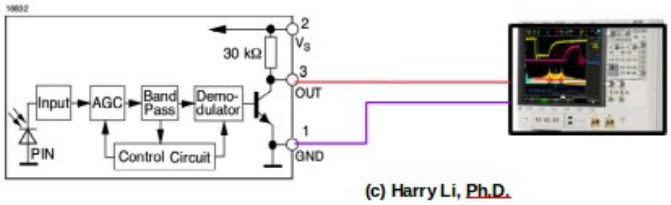
The TSOP12xx - series are miniaturized receivers for infrared remote control systems.





Reference: TSOP1236 datasheet

Example: Measure TSOP1236 response to the color remote controller's signal



(c) Harry Li, Ph.D.

INT Special Purpose Registers for Init

(1) function prototype: `uint32_t EINTInit( void )` the type is uint32_t, unsigned integer, right click on it to check its definition, this bring you to stdint.h header file, so you can see *typedef* unsigned *int uint32_t;*

(2) Special purpose registers (6):

PINSEL4;

PINMODE4;

IO2IntEnR; IO2IntEnF; Table 116. Interrupt Enable for port 2 Rising Edge (IO2IntEnR)

EXTMODE; EXTPOLAR; Table 11. External Interrupt Mode register (EXTMODE)

Table 12. External Interrupt Polarity register EXTPOLAR

```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;            // for making pull-up use 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11)); /* Port2.11 is rising edge. */
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<11));
 LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE ;      /* INT1 edge trigger */
 LPC_SC->EXTPOLAR |= 0;                  /* INT0 is falling edge by default */
 NVIC_EnableIRQ(EINT1_IRQn);
 return 0;
}
```

(c) Harry Li, Ph.D.

Pin Function Select Register PINSEL4

Example:

```
LPC_PINCON->PINSEL4 &= ~(3 << 22 );
//set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
```

Table 84: Pin function select register 4 (PINSEL4), pp 119.

PINSEL4	Pin name	Function when 00	Function when 01
1:0	P2.0	GPIO Port 2.0	PWM1.1
3:2	P2.1	GPIO Port 2.1	PWM1.2
5:4	P2.2	GPIO Port 2.2	PWM1.3
7:6	P2.3	GPIO Port 2.3	PWM1.4
9:8	P2.4	GPIO Port 2.4	PWM1.5
11:10	P2.5	GPIO Port 2.5	PWM1.6
13:12	P2.6	GPIO Port 2.6	PCAP1.0
15:14	P2.7	GPIO Port 2.7	RD2
17:16	P2.8	GPIO Port 2.8	TD2
19:18	P2.9	GPIO Port 2.9	USB_CONNECT
21:20	P2.10	GPIO Port 2.10	$\overline{\text{EINT0}}$
23:22	P2.11[1]	GPIO Port 2.11	$\overline{\text{EINT1}}$
25:24	P2.12[1]	GPIO Port 2.12	$\overline{\text{EINT2}}$

Pin Mode Select Register PINMODE

The PINMODE registers control the input mode of all ports. This includes the use of the on-chip pull-up/pull-down resistor feature and a special open drain mode. pp. 114

Example:

```
LPC_PINCON->PINMODE4 = 0;
```

Table 77. Pin Mode Select register Bits, pp.

PINMODE0 to PINMODE9 Values	Function
00	Pin has an on-chip pull-up resistor enabled.
01	Repeater mode (see text below).
10	Pin has neither pull-up nor pull-down resistor enabled.
11	Pin has an on-chip pull-down resistor enabled.

Table 92. Pin Mode select register 4: PINMODE4, pp. 123

PINMODE4	Symbol	Description
1:0	P2.00MODE	Port 2 pin 0 control, see P0.00MODE.
3:2	P2.01MODE	Port 2 pin 1 control, see P0.00MODE.
5:4	P2.02MODE	Port 2 pin 2 control, see P0.00MODE.
7:6	P2.03MODE	Port 2 pin 3 control, see P0.00MODE.
9:8	P2.04MODE	Port 2 pin 4 control, see P0.00MODE.
11:10	P2.05MODE	Port 2 pin 5 control, see P0.00MODE.
13:12	P2.06MODE	Port 2 pin 6 control, see P0.00MODE.
15:14	P2.07MODE	Port 2 pin 7 control, see P0.00MODE.
17:16	P2.08MODE	Port 2 pin 8 control, see P0.00MODE.
19:18	P2.09MODE	Port 2 pin 9 control, see P0.00MODE.

Interrupt Enable for P2 Rising Edge IO2IntEnR

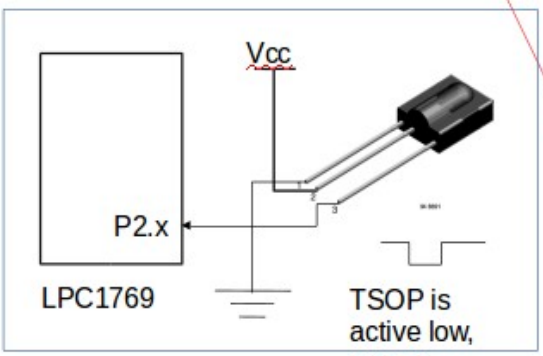
Table 116. GPIO Interrupt Enable for port 2 Rising Edge (IO2IntEnR), chapter 9, pp. 141

6	P2.6ER	Enable rising edge interrupt for P2.6.
7	P2.7ER	Enable rising edge interrupt for P2.7.
8	P2.8ER	Enable rising edge interrupt for P2.8.
9	P2.9ER	Enable rising edge interrupt for P2.9.
10	P2.10ER	Enable rising edge interrupt for P2.10.
11	P2.11ER[1]	Enable rising edge interrupt for P2.11.

Example:

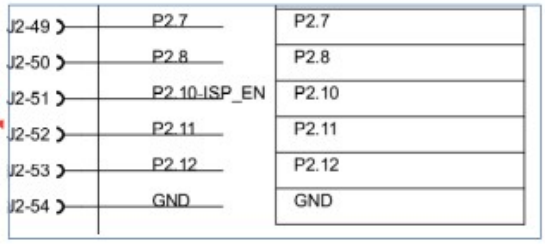
```
LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11));
```

For the falling edge interrupt, see 9.5.6.5 GPIO Interrupt Enable for port 2 Falling Edge (IO2IntEnF), pp. 143.



TSOP is active low, so use IO2IntEnF

Pin assignment for P2, from schematics of the LPC1769 module



EXTMODE: The External Interrupt Mode Register

The External Interrupt Mode Register controls whether each pin is edge- or level-sensitive. pp. 35 from CPU data sheet.

Example:

```
LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE
```

Table 11. External Interrupt Mode register (EXTMODE)

Bit	Symbol	Value	Description
0	EXTMODE0	0	Level-sensitivity is selected for $\overline{\text{EINT0}}$.
		1	$\overline{\text{EINT0}}$ is edge sensitive.
1	EXTMODE1	0	Level-sensitivity is selected for $\overline{\text{EINT1}}$.
		1	$\overline{\text{EINT1}}$ is edge sensitive.
2	EXTMODE2	0	Level-sensitivity is selected for $\overline{\text{EINT2}}$.
		1	$\overline{\text{EINT2}}$ is edge sensitive.

Note:

Edge triggered (sensitive) or level triggered (sensitive) detect the interrupt signal based on its level or its (raising or falling) edge. Edge triggered are more timing sensitive to capture the sudden change of the signal, however, random noise can easily mis-triggered the interrupt then level triggered interrupt.

External Interrupt Polarity Register EXTPOLAR

In level-sensitive mode, the bits in this register select whether the corresponding pin is high- or low-active. In edge-sensitive mode, they select whether the pin is rising- or falling-edge sensitive. pp.27 data sheet.

Example:

```
LPC_SC->EXTPOLAR |= 0;
```

Note the above code sets falling edge or level interrupt from NXP sample code, but the following could be better

```
LPC_SC->EXTPOLAR &= ~(1<<2);
```

Table 12. External Interrupt Polarity register EXTPOLAR

Bit	Symbol	Value	Description
0	EXTPOLAR0	0	$\overline{\text{EINT0}}$ is low-active or falling-edge sensitive (depending on EXTMODE0).
		1	$\overline{\text{EINT0}}$ is high-active or rising-edge sensitive (depending on EXTMODE0).
1	EXTPOLAR1	0	$\overline{\text{EINT1}}$ is low-active or falling-edge sensitive (depending on EXTMODE1).
		1	$\overline{\text{EINT1}}$ is high-active or rising-edge sensitive (depending on EXTMODE1).
2	EXTPOLAR2	0	$\overline{\text{EINT2}}$ is low-active or falling-edge sensitive (depending on EXTMODE2).
		1	$\overline{\text{EINT2}}$ is high-active or rising-edge sensitive (depending on EXTMODE2).

NVIC_EnableIRQ(*EINT1_IRQn*)

```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;              // for making pull-up use 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11)); /* Port2.11 is rising edge. */
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<11));
 LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE ;      /* INT1 edge trigger */
 LPC_SC->EXTPOLAR |= 0;              /* INT0 is falling edge by default */
 NVIC_EnableIRQ(EINT1_IRQn);
 return 0;
}
```

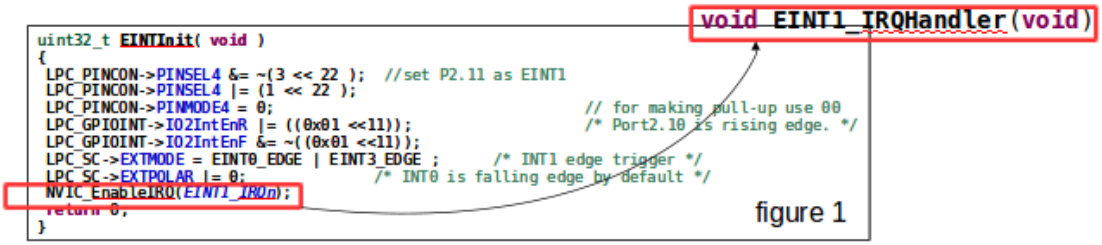
Need to use ExINT in the design, from CPU data sheet, the ExINT maps to GPP port 2, e.g., P2.x

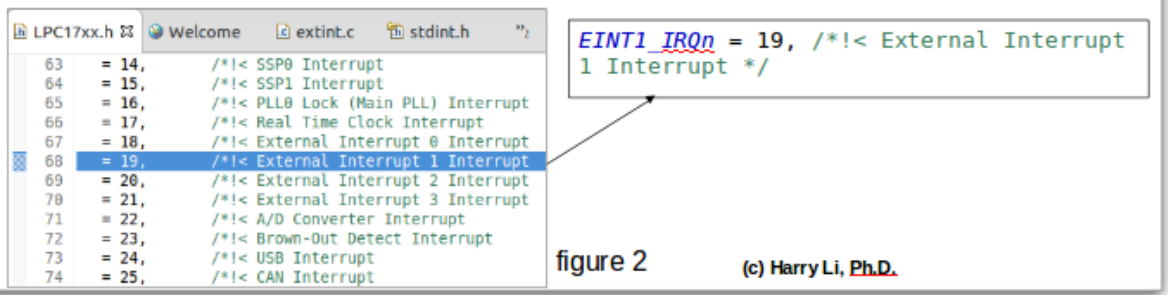
Then to find the pins of P2.x, from the CPU data sheet, table 84, Pin Function Select Register 4 (PINSEL4), pp. 119.

15:14	P2.7	GPIO Port 2.7	RD2
17:16	P2.8	GPIO Port 2.8	TD2
19:18	P2.9	GPIO Port 2.9	USB_CONNECT
21:20	P2.10	GPIO Port 2.10	$\overline{\text{EINT0}}$
23:22	P2.11[1]	GPIO Port 2.11	$\overline{\text{EINT1}}$
25:24	P2.12[1]	GPIO Port 2.12	$\overline{\text{EINT2}}$
27:26	P2.13[1]	GPIO Port 2.13	$\overline{\text{EINT3}}$

Interrupt Number Linked to ISR

Interrupt Service Routine (ISR): *NVIC_EnableIRQ(EINT1_IRQn);*
(1) where is EINT1_IRQn is declared? LPC17xx.h;
Mouse over on it (fig 1) then click on open declaration, pop-up window shows EINT1_IRQn at /CMSIS_CORE_LPC17xx/inc/LPC17xx.h
Check its declaration details, click on its item on the pop-up window, see its declaration, (fig 2); (2) How is ISR connected?





ISR Example

After initialization, now use interrupt. Sample code touch button switch to turn on/off LED based on interrupt is given below.

```
void EINT1_IRQHandler (void){
LPC_SC->EXTINT = EINT1;

LPC_GPIO0->FIODIR |= (1<<3);
if(LPC_GPIO2->FIOPIN &(1<<11)){
key_count ++;  // key_count +1 when receive external interrupt
    delayMs(0,500);  //delay used as debouncer
}
if( key_count == (key_count1 + key_count2)){
if(LPC_GPIO0->FIOPIN & (1<<3))
{
   LPC_GPIO0->FIOCLR |= (1<<3);  //turn off led if it was on
}
else
LPC_GPIO0->FIOSET |= (1<<3);    //turn on led if it was off
key_count1 += key_count2;
key_count2 ++;                  //increment count calculate the number of
times for on and off led //press button once to turn on LED, twice to turn
off, 3 times to turn on LED  //4 times to turn off LED ...
}
LPC_GPIOINT->IO2IntEnR = ((0x01 <<11));
LPC_GPIOINT->IO2IntClr = 0xFFFFFFFF;
LPC_GPIOINT->IO0IntClr = 0xFFFFFFFF;
}
```

Interrupt Set Enable Register to ISR

Interrupt Set Enable Register is is located via interrupt number (type), e.g., *NVIC_EnableIRQ(EINT1_IRQn) (defined* in core_cms3.h see the sample code below) which is in turn mapped to interrupt table in the memory map, at the corresponding location of this table holds the pointer pointing to *Service Routine (ISR)*

(1)enable device specific interrupt;

(2) the interrupt controller is *NVIC;*

```
/* Enable External Interrupt. The function enables a device-specific
interrupt in the NVIC interrupt controller. param [in] IRQn External INT
number. Value cannot be negative. */
__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F));
  /* enable interrupt */
}
```

figure 1

Putting INIT and ISR Together

Putting INI (initialization), and user defined ISR together .

```
int main(void)
{

LPC_GPIO0->FIODIR |= (1<<3); //set pin 0.23 as
output
LPC_GPIO0->FIODIR &= ~(1<<11); //set pin 2.11
as input
LPC_GPIO0->FIOCLR |= (1<<3); //clear pin 0.23

while(1)
{
EINTInit();   //wait for external interrupt
}
```

```
uint32_t EINTInit( void )
{
 LPC_PINCON->PINSEL4 &= ~(3 << 22 );  //set P2.11 as
EINT1
 LPC_PINCON->PINSEL4 |= (1 << 22 );
 LPC_PINCON->PINMODE4 = 0;    //making pull-up 00
 LPC_GPIOINT->IO2IntEnR |= ((0x01 <<11));    //Port2.10
rising edge
 LPC_GPIOINT->IO2IntEnF &= ~((0x01 <<11));
 LPC_SC->EXTMODE = EINT0_EDGE | EINT3_EDGE;//INT1 edge
trigger
 LPC_SC->EXTPOLAR |= 0;        // INT0 is falling edge
by default
 NVIC_EnableIRQ(EINT1_IRQn);
 return 0;
}
```

```
void EINT1_IRQHandler (void)
{
LPC_SC->EXTINT = EINT1;
LPC_GPIO0->FIODIR |= (1<<3);
if(LPC_GPIO2->FIOPIN &(1<<11))
{
key_count ++;  // key_count +1 when receive
interrupt
    delayMs(0,500);  //delay used as
debouncer
}
if( key_count == (key_count1 + key_count2))
{
if(LPC_GPIO0->FIOPIN & (1<<3))
{
   LPC_GPIO0->FIOCLR |= (1<<3);  //turn off
led if on
}
else
LPC_GPIO0->FIOSET |= (1<<3);    //turn on
led if off
key_count1 += key_count2;
key_count2 ++;                  //increment
the count
}
LPC_GPIOINT->IO2IntEnR = ((0x01 <<11));
LPC_GPIOINT->IO2IntClr = 0xFFFFFFFF;
LPC_GPIOINT->IO0IntClr = 0xFFFFFFFF;
}
```

INT Timer Technique

Power Up Timers

(1) Power up timer

1. Power: In the PCONP register (Table 46), set bits PCTIM0/1/2/3. On reset, Timer0/1 are enabled (PCTIM0/1 = 1), and Timer2/3 are disabled (PCTIM2/3 = 0). Chapter 21, pp 499;

2. Table 46. Power Control for Peripherals register (PCONP - address 0x400F C0C4)

```
int main(void)
{
/* Power up timer0 */
LPC_SC->PCONP |= BIT(1);
/* Clear all interrupts */
LPC_TIM0->IR = 0x3F;
/* Set prescaler based on sysclk to get
1000us minimum interval */
switch (PCLKSEL_TIMER0(LPC_SC-
>PCLKSEL0)) {
case 0x00:
LPC_TIM0->PR = SystemCoreClock / 4000;
break;
```

Example: based on table 46, power up Timer 1, write a C-code to realize this task

Bit	Symbol	Description
0	-	Reserved.
1	PCTIM0	Timer/Counter 0 power/clock control bit.
2	PCTIM1	Timer/Counter 1 power/clock control bit.
3	PCUART0	UART0 power/clock control bit.

```
/* Power up timer1 */
LPC_SC->PCONP |= BIT(2);
```

Or

```
LPC_SC->PCONP |= 0x4;
```

Clear All Timer INTs

(2) `LPC_TIM0->IR` Interrupt Register; (3) Set Prescaler

Table 427. Interrupt Register - addresses 0x4000 4000, IR consists of 4 bits for the match interrupts, if interrupt generated then its corresponding bit will set high. chapter 21, pp. 502; you will need to clear it for the next interrupt.

Example: based on table 427, clear TIM3->IR, write a C-code to realize this task

```
int main(void)
{
/* Power up timer0 */
LPC_SC->PCONP |= BIT(1);
/* Clear all interrupts */
LPC_TIM0->IR = 0x3F;
/* Set prescaler based on sysclk to get
1000us minimum interval */
switch (PCLKSEL_TIMER0(LPC_SC-
>PCLKSEL0)) {
case 0x00:
LPC_TIM0->PR = SystemCoreClock / 4000;
break;
}
```

Bit	Symbol	Description
0	MR0 Interrupt	Interrupt flag for match channel 0.
1	MR1 Interrupt	Interrupt flag for match channel 1.
2	MR2 Interrupt	Interrupt flag for match channel 2.
3	MR3 Interrupt	Interrupt flag for match channel 3.

```
LPC_TIM0->IR |= 0x4;
```

How to clear both Timer 3 and 4? Can you change the above c code to the following for the same clear function?

```
LPC_TIM0->IR = 0x4;
```

Fig. 1

Timing Waveforms and Prescaler PR

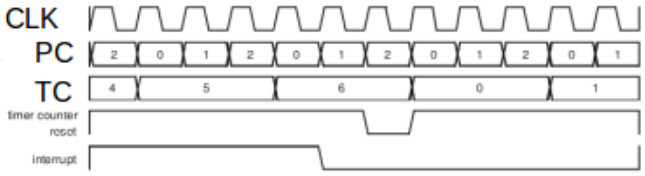
(3) Set Prescaler

Note: 4 important registers, PR, PC, TC, and MR

Two steps: first, `PCLKSEL register, Peripheral Clock Selection register, selects Timer0 clock`; Section `4.7.3, pp. 57;`

2nd, `LPC_TIM0->PR` Prescaler register, When the Prescale Counter (PC) is equal to PR, Timer Counter (TC) is incremented by 1;
e.g,
TC is incremented every PR+1 cycles of PCLK.

Bit	Symbol	Description
1:0	PCLK_WDT	Peripheral clock selection for WDT.
3:2	PCLK_TIMER0	Peripheral clock selection for TIMER0.
5:4	PCLK_TIMER1	Peripheral clock selection for TIMER1.




```
int main(void)
{
/* Power up timer0 */
LPC_SC->PCONP |= BIT(1);
/* Clear all interrupts */
LPC_TIM0->IR = 0x3F;
/* Set prescaler based on sysclk to
get 1000us minimum interval */
switch (PCLKSEL_TIMER0(LPC_SC-
>PCLKSEL0)) {
case 0x00:
LPC_TIM0->PR = SystemCoreClock / 4000;
```

Example:
Given PR = 2, MR = 6; An interrupt generated on match.
(1) at every clock, PC -> PC+1;
(2) when PC = PR, TC -> TC+1 ;
(3) when TC = MR, INT generated.

Set MR, MCR and TCR

(4) set match register MR and match control register MCR

```
/* Set the interval at Match Control 0 */
LPC_TIM0->MR0 = TIMER_INTERVAL;
/* Enable Match Control 0 */
LPC_TIM0->MCR = BIT(0) | BIT(1);
/* Disable Timer as default */
LPC_TIM0->TCR = 0;
/* Enable Timer IRQ */
NVIC_EnableIRQ(TIMER0_IRQn);
/* Set P2.10 as EINT0 function */
LPC_PINCON->PINSEL4 &= ~GEN_MASK(2, 20);
LPC_PINCON->PINSEL4 |= BIT(20);
/* Enable falling edge trigger for P2.10 */
LPC_GPIOINT->IO2IntEnF |= BIT(10);
/* Edge trigger for EINT0 */
LPC_SC->EXTMODE |= BIT(0);
/* Enable EINT IRQ */
NVIC_EnableIRQ(EINT0_IRQn);

printf("Main Function: Waiting for
interrupt\n");

    while(1);

    return 0;
}
```

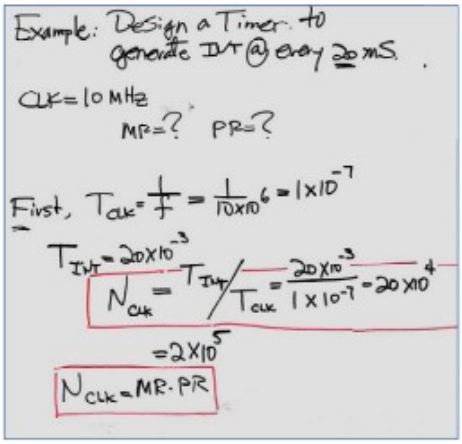
MR values are continuously compared to the Timer Counter value. When matched, action takes place as one of the following three: an interrupt, reset the Timer Counter, or stop the timer. Actions controlled by the settings in the MCR register.

Table 430 (pp. 505). Match Control Register (T[0/1/2/3]MCR - addresses 0x4000 4014

Bit	Symbol	Value	Description
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.
		0	This interrupt is disabled
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.
		0	Feature disabled.
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.
		0	Feature disabled.

Timer Counter registers (T0TC – T3TC) is incremented when the prescale counter reaches its terminal count. Chapter 21, pp. 504.

Design Example



SO, let

PR = 200;
MR = 1000;

PR and MR both are 32 bit registers, see data sheet, pp. 504 (PR) and 505 (MR)

21.6.5 Prescale register (T0PR - T3PR)
The 32-bit Prescale register specifies the maximum value for the Prescale Counter.

V. Two Dimensional Graphics Process Engine Design and Implementation

5.1 Design Requirement

This lab counts total 10 points. Note hard copy of the lab report has to be ready for the time to make in class demo, fail to bring the hard copy can result in 5 marks reduction. In addition, the soft copy of the report plus the source code exported as a project have to be submitted on line. This lab as a preliminary step to build 2D vector graphics processing engine, you will

1. Design and prototype LPC1769 micro-processor system board, and enable one SPI LCD display by designated LPC1769 node as your choice. It can be either slave or master, which can be defined in the later labs. Let's define this LPC1769 as LPC1769 LCD node.

2. Use draw line test code to run draw-a-line testing program from you LPC1769-LCD node. Once this is done, you are ready for the implementation of 2D screen saver functions 1 and 2. Function 1 is for rotating square patterns display and Function 2 is for generating trees.

3. Generate 2D screen saver of rotating squares based on vector graphics formula discussed in the class (5 points)

(1) use the following equation:

P(x,y) = P_1(x_1,y_1) + lamda * (P_2(x_2,y_2) - P_1(x_1,y_1))

with lamda = 0.8 by default, and lamda = 0.2 when prompted for user selected input;

(2) create two dimensional rotating patterns with data set of "parent" square;

(3) randomized location by using rand() function;

(4) randomized reduction of the parent square;

(5) choose one color for each set of rotation patterns, and rotates at least 10 levles or higher;

(6) continue to display each set of patterns without erasing the patterns.

4. Generate 2D trees with its branches level no less than 10 or higher based on vector graphics formula discussed in the class (5 points)

(1) use P(x,y) = P_1(x_1,y_1) + lamda * (P_2(x_2,y_2) - P_1(x_1,y_1)) with lamda = 0.8 by default for tree branch reduction;

(2) create patch of forest by modifying one parent tree;

(3) randomized location of the new trees by using rand() function;

(4) randomized reduction of the parent tree trunks and branches;

(5) randomized angles for the branches;

(6) continue to display trees without erasing till the keyboard input detected.

5. Submit project report together with

(1) exported project (source code) subject to testing and verification, including compilation and build, as well as actual LPC1769 board testing.

5.2. Assessment

(1) Satisfies the requirements stated in 2018S-17-Lab-report-rubrics.txt;

(2) lab report should cover (hardware side):

(2.1) system block diagrams of the entire system setup including laptop computer;

(2.2) system block diagram of the SPI color LCD interface;

(2.3) Schematics of the LPC1769 interface to LCD color display panel;

(2.4) table(s) of the pin connectivity;

(2.5) photo(s) of the implementation.

(3) lab report:

(3.1) software part should cover

a. Algorithm description;

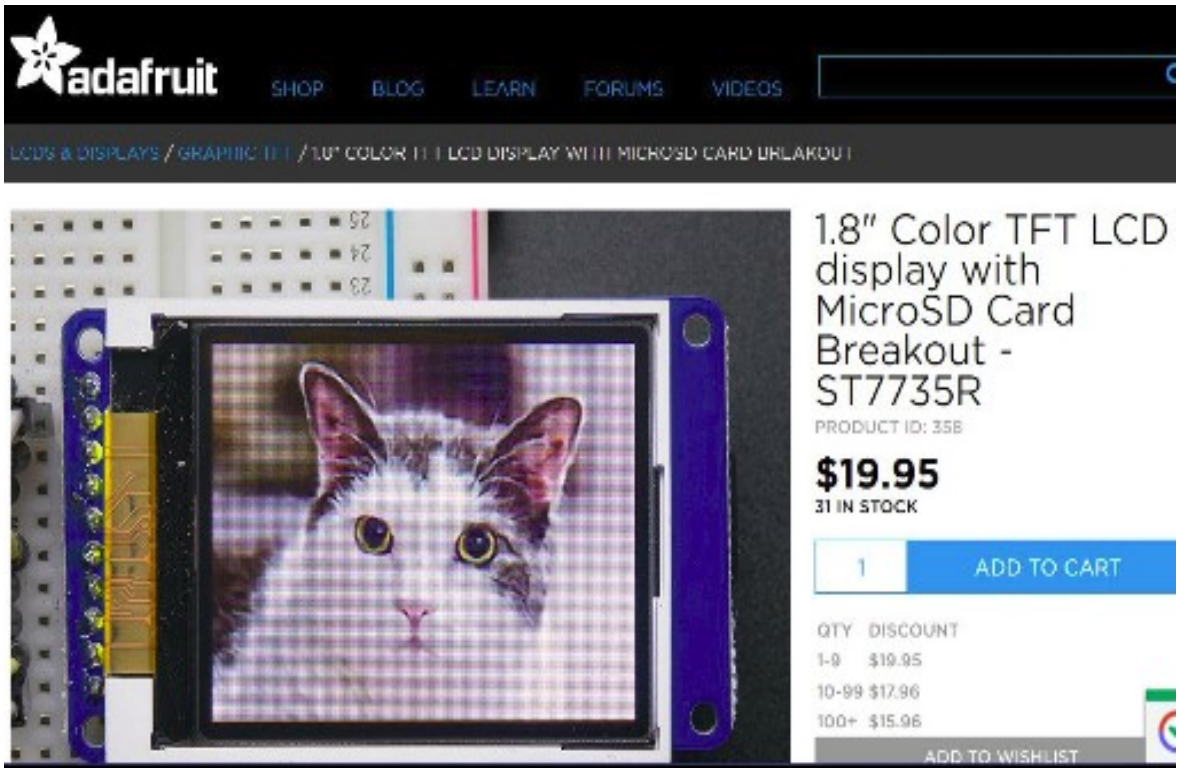
b. Flow chart(s);

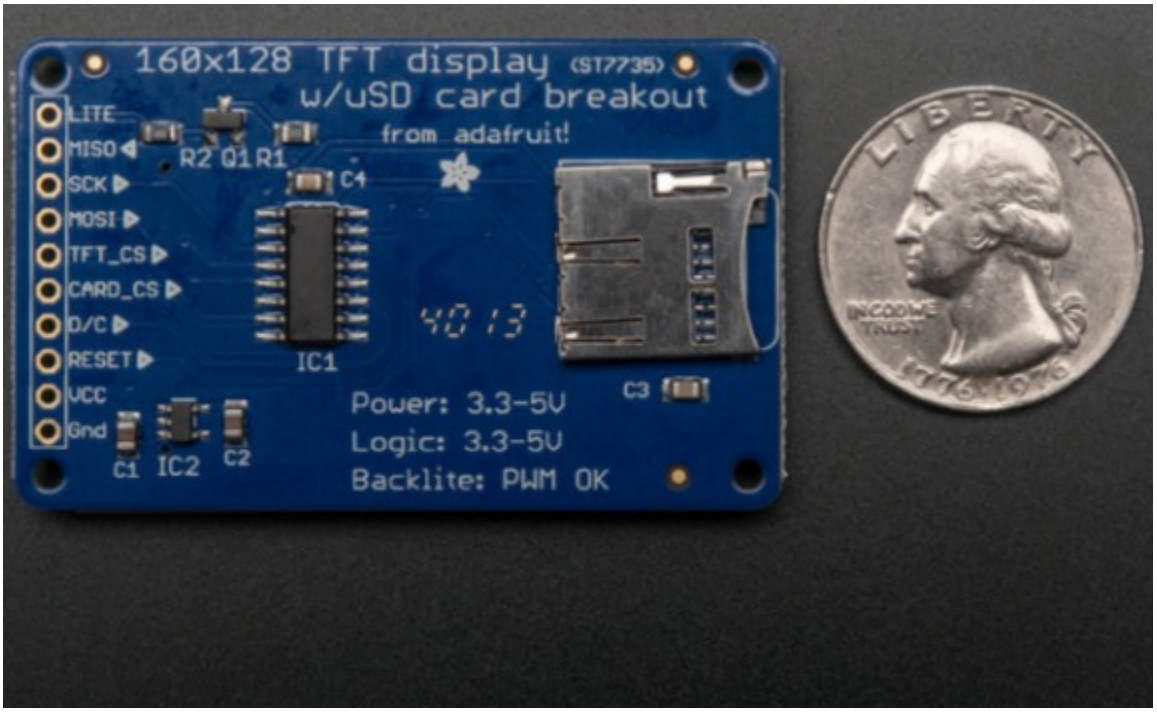
c. Pseudo code;

d. testing and verification section;

e. source code listing (appendix).

5.3. Design Notes





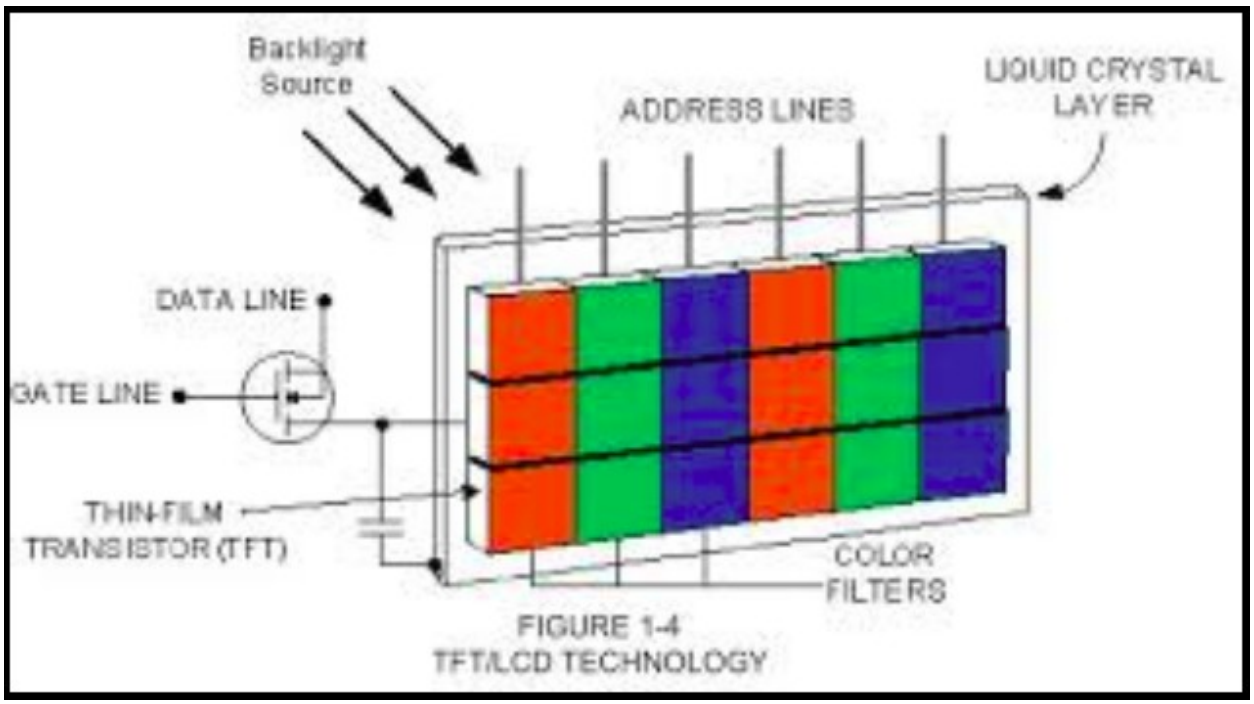


FIGURE 1-4
TFT/LCD TECHNOLOGY

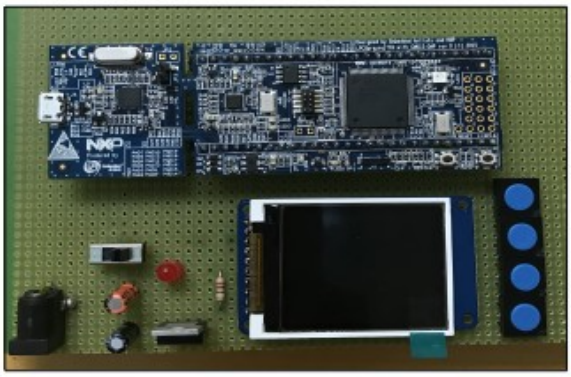
LPC1769 DISPLAY UNIT

- LPC 1769
- Color TFT LCD display

Resolution : 128x160,
Pixel Depth: 18-bit (262,144) colors
Controller: ST7735
Interface: SPI interface

LCD Pins	LPC1769 Pins
Gnd	Gnd
VCC	VCC (3.3V)
RST	GPIO output (P0.22)
RS/DC	GPIO output (P0.21)
CARD_CS	X
TFT_CS	SSEL0 (P0.16)
MOSI	SSP0 MOSI (P0.18)
SCK	SCK0 (P0.15)
MISO	SSP0 MISO (P0.17)
LITE	VCC (3.3V)

Note from Harry Li: if for new SPI LCD module, use this pin

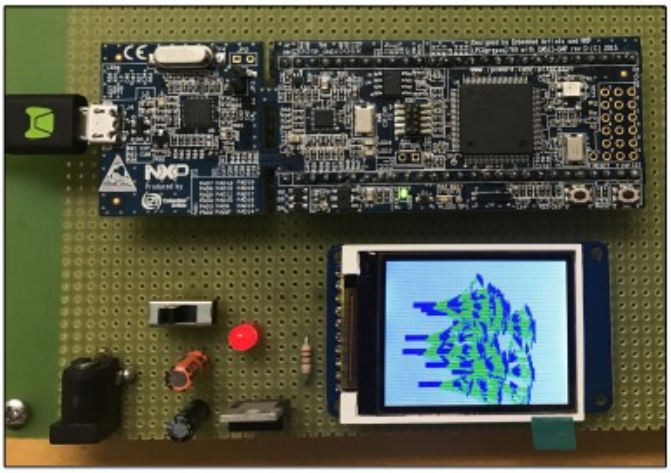


Note from Harry Li: for the class, ignore 4 button switch, just connect SPI LCD.

LPC-1769		LCD
Label	Pin	
MOSI	P 0.18	SDA
MISO	P0.17	MISO
SCK	P0.15	SCL
CS	P0.16	TFT_CS
GPIO-DC (Data / command)	P0.21	AO
GPIO-Reset	P0.22	RST
3.3V		LED+
GND		LED-

TEST LCD DISPLAY

- Build Project Lcd_Test and check for any errors
- Debug Lcd_Test and run the program
- Fractal tree is drawn on the LCD display



9-17-2018 SPI LCD Hardware I/F

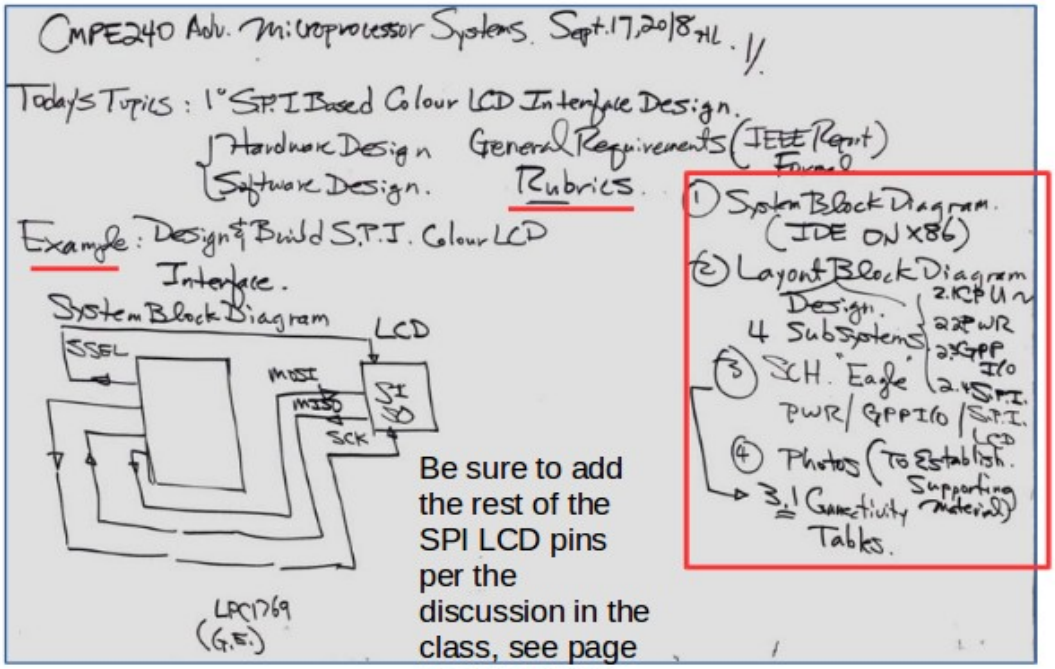
CMPE240 Adv. Microprocessor Systems. Sept. 17, 2018.

Table 1. Connectivity B/W LPC and Colour LCD.

CPU	Colour LCD	Note
MOSI/MOSI1/P0.9/J2-5	SI/"MOSI"	(Pin Number from LCD).
MISO/MISO1/P0.8/J2-6	SO	
SCK/SCK1/P0.7/J2-7	SCK(I)/SCK	
SSEL/SSEL1/P0.6/J2-8	ENABLE/TFT_CS	
Vcc/GPP/P0.28./J2-26.	LIGHT LCD	
Data/S.Ctrl/GPP/P0.21/J2-23	Data/Command	
Reset/GPP/P0.22/J2-24	RESET	
VID_3V3X	Vcc(3.3VDC)	
GND	GND.	

Check your SPI LCD module, different version may have slightly different pin connectivity, use the table on page 1 from this PPT for the newer version

9-17-2018 SPI Hardware And Rubrics



Be sure to add the rest of the SPI LCD pins per the discussion in the class, see page 1 connectivity table on this PPT

SSP0 CONFIGURATION

- Set PCSSP0 interface power/clock bit to one in the PCONP register
- Set Peripheral clock selections PCLK_SSP0 bits in PCLKSEL1 register
- Select Pins P0.15 – P0.18 through PINSEL registers and PINMODE
- Select data size, frame format, CPOL,CPHA and SCR in the control register SSP0CR0
- SSP0CPSR, clock prescale register is set
- Enable interrupt in NVIC using SSP0_IRQn register
- Enable master mode on the SSP by setting SSE bit in SSP0CR1 register to one
- Set RORIM and RTIM bits to 1 in SSP0IMSC register to enable error related interrupts

Main Code

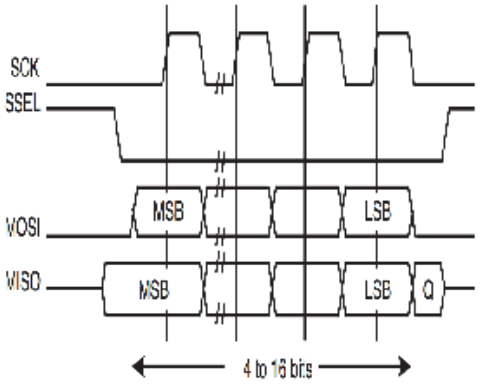
```
int main (void)
{
        uint32_t pnum = PORT_NUM;
        pnum = 0 ;
        if ( pnum == 0 )
                SSP0Init();
        else
                puts("Port number is not correct");
        lcd_init();

        fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, WHITE);
        int x0,x1,y0,y1;

        x0 = 20;
        x1 = 80;
        y0 = 60;
        y1 = 140;

        drawLine(x0,y0,x1,y1,PURPLE);
        return 0;
}
```

Reference: CTI One Corporation

LCD DRIVERS

Send command/data to the LCD controller using SSP0 port

```
void spiwrite(uint8_t c)

{

 int pnum = 0;

 src_addr[0] = c;

 SSP_SSELToggle( pnum, 0 );

 SSPSend( pnum, (uint8_t *)src_addr, 1 );

 SSP_SSELToggle( pnum, 1 );

}
```



- To start data transmission, SSEL0 master signal is driven to low.
- Data is captured at each clock cycle.
- SSEL0 master signal is changed to high to indicate end of data transmission

LCD DRIVERS

Send command/data to the LCD controller using SSP0 port

```
void writecommand(uint8_t c)
{
 LPC_GPIO0->FIOCLR |= (0x1<<21);
 spiwrite(c);
}
```

```
void writedata(uint8_t c)
{
 LPC_GPIO0->FIOSET |= (0x1<<21);
 spiwrite(c);
}
```

```
void writeword(uint16_t c)
{
 uint8_t d;
 d = c >> 8;
 writedata(d);
 d = c & 0xFF;
 writedata(d);
}
```

- Serial information sent to LCD can be either be data or command
- For Command, the LCD's RS/DC input (Pin 0.21 on LPC1769) must be 0
- For Data, the LCD's RS/DC input (Pin 0.21 on LPC1769) must be 1
- Information is sent on the SSP0 port after setting or clearing Pin 0.21
- writeword function transfers 2 bytes (16 bits) of data

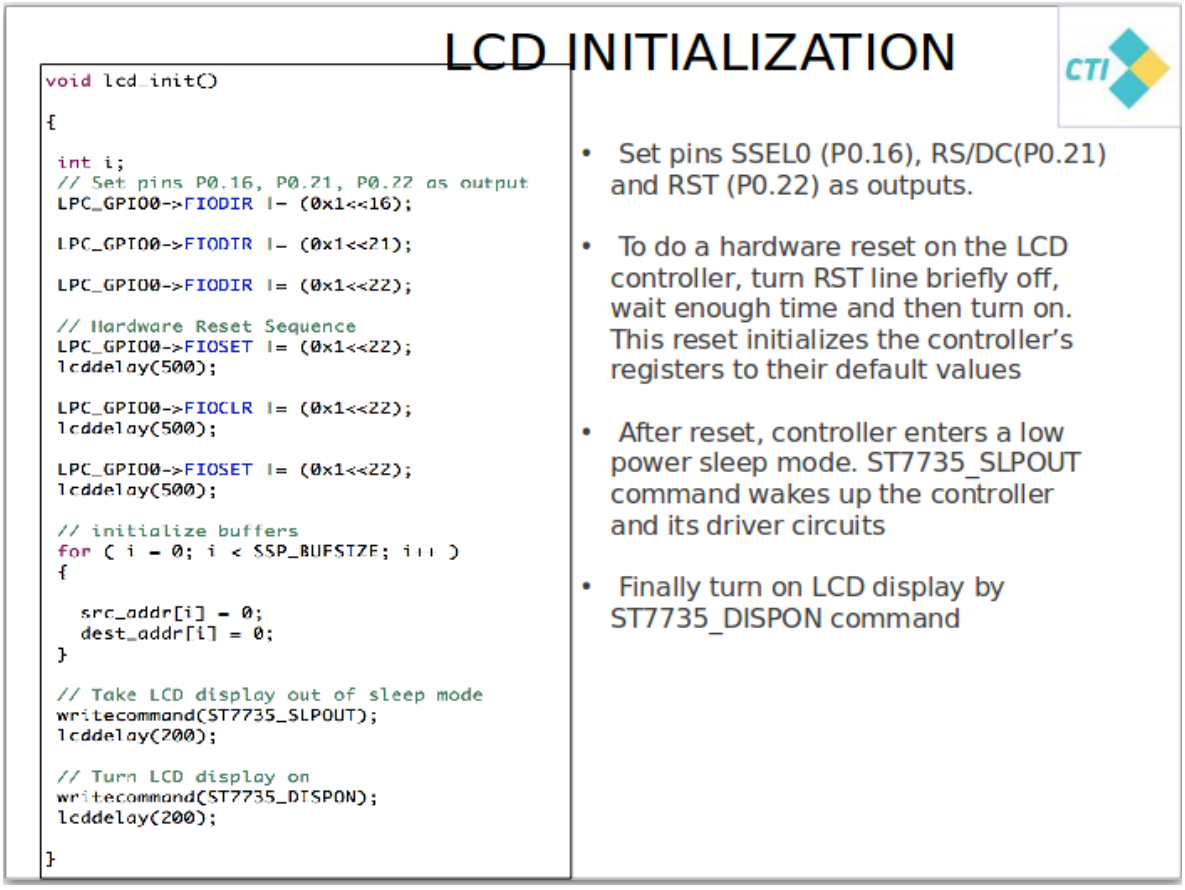
LCD DRIVERS

Write color data on the LCD display

```
void write888(uint32_t color, uint32_t repeat)
{
 uint8_t red, green, blue;
 int i;
 red = (color >> 16);
 green = (color >> 8) & 0xFF;
 blue = color & 0xFF;
 for (i = 0; i< repeat; i++) {
  writedata(red);
  writedata(green);
  writedata(blue);
 }
```

- Pixel colors are a combination of Red, Green and Blue colors.
- Each sub color is represented by 8 bits.

Color	(R,G,B)	Hex
Red	(255,0,0)	#FF0000
Green	(0,255,0)	#00FF00
Blue	(0,0,255)	#0000FF
Black	(0,0,0)	#000000
White	(255,255, 255)	#FFFFFF



VI. AGV Series Product Motor Controlling Functions

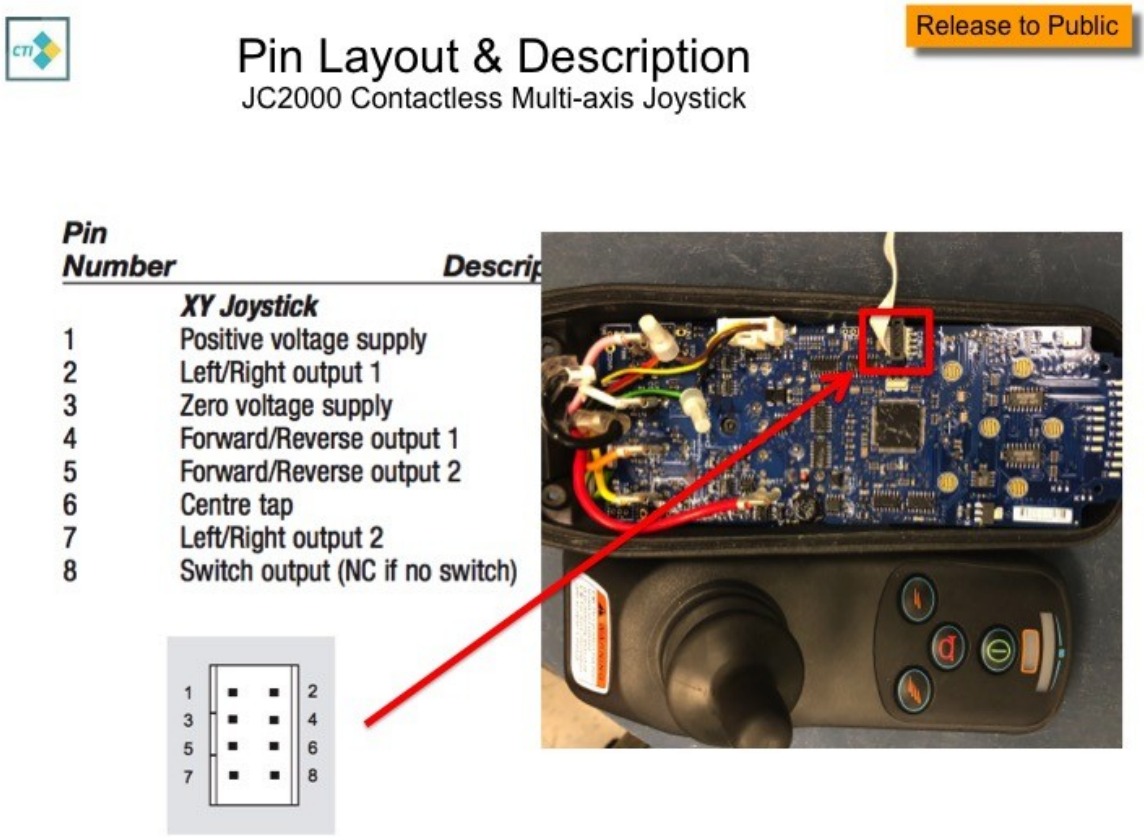
6.1 Design Requirement

This lab counts total 10 points. Note hard copy of the lab report has to be ready for the time to make in class demo, fail to bring the hard copy can result in 5 marks reduction. In addition, the softcopy of the report including the source code exported as a project have to be submitted online. This lab is a preliminary step to use a microprocessor to control motor drive, you will

1. Getting familiar with NXP17xx module OM13085, which is a Cortex-M3 microcontroller for embedded applications featuring a high level of integration. Understanding all the features it has including 512 kB of flash memory, 64 kB of data memory, Ethernet MAC, USB Device/Host/OTG, 8-channel DMA controller, 4

UARTs, 2 CAN channels, 3 SSP/SPI, 3 I2C, I2S, 8-channel 12-bit ADC, 10-bit DAC, motor control PWM, Quadrature Encoder interface, 4 general purpose timers, 6-output general purpose PWM, ultra-low power Real-Time Clock with separate battery supply, and up to 70 general purpose I/O pins.

2. Completing data collection of all pins from JC2000 Contactless Multi-axis driver on the electrical wheel chair. You need to create a table to find the range of input and output voltage signal from all 8 pins on the driver board to analysis the required signal that is able to control the vehicle.
3. Designing and building circuits to make LPC17xx module fit the driver board based on the voltage signal range. At the same time, integrating PWM and GPIO digital signals to simulate all the control pins output voltage to the motor drive board in order to perform all the driving functions.



Testing Data Table

JC2000 Contactless Multi-axis Joystick

Release to Public

	Description	Original	Forward	Reverse	Right	Left	Forward Right	Forward Left
#1	Positive Voltage (Input)	5	4.92	4.92	4.92	4.92	4.92	4.92
#2	Left/Right 1 (output)	2.45	2.47	2.5	1.08	3.75	1.61	3.35
#3	Zero Voltage (output)	GND	GND	GND	GND	GND	GND	GND
#4	Forward/ Reverse 1 (output)	2.45	3.78	1.06	2.32	2.05	3.40	3.28
#5	Forward/ Reverse 2 (output)	2.45	3.77	1.03	2.45	2.00	3.09	3.15
#6	Centre Tap (output)	2.45	2.44	2.45	2.44	2.44	2.44	2.45
#7	Left/Right 2 (output)	2.44	2.40	2.44	1.05	3.67	1.30	3.42
#8	Switch (output)	0	0	0	0	0	0	0

Testing Data Analysis

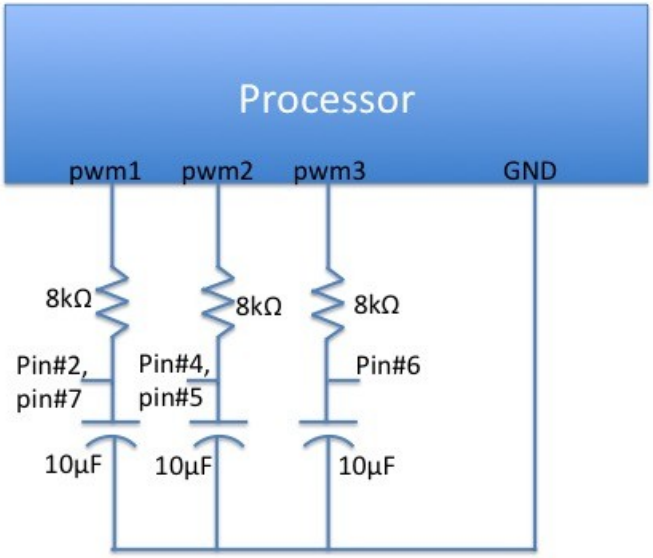
JC2000 Contactless Multi-axis Joystick

Release to Public

	Right Related	**Left Related**	**Forward Related**	**Reverse Related**
#1	Input to Joystick, Always 5v			
#3	Always GND			
#2 & #7	Decreasing, minimum 1.08v	Increasing, maximum 3.75v	No change, stay 2.45v	No change, stay 2.45v
#4 & #5	No change, stay 2.45v	No change, stay 2.45v	Increasing, maximum 3.78v	Decreasing, minimum 1.03v
#6	Output from Joystick, Always 2.45v			
#8	Always 0v			

Release to Public

PWM Output Control



Release to Public

PWM Control Table

Frequency: 2kHz
Duty cycle range: 55% to 95%

Output voltage from processor to motor control board.

	Straight Forward	Right Forward	Left Forward	Straight Reverse	Right Reverse	Left Reverse	Stop
Pin #2 & Pin #7	2.45V	1.08V to 2.2V	2.8V to 3.75V	2.45V	1.08V to 2.2V	2.8V to 3.75V	2.45V
Pin #4 & Pin #5	2.8V to 3.75V	2.8V to 3.75V	2.8V to 3.75V	1.08V to 2.2V	1.08V to 2.2V	1.08V to 2.2V	2.45V
Pin #6	2.45V	2.45V	2.45V	2.45V	2.45V	2.45V	2.45V
Pin #3	GND	GND	GND	GND	GND	GND	GND

Release to Public

LPCNOD PWM Control

P2.0 (J2-42)
Controls Left/Right

P2.1 (J2-43)
Controls Forward/Reverse

P2.2 (J2-44)
Constant Output 2.44v for
Centre Tap Pin

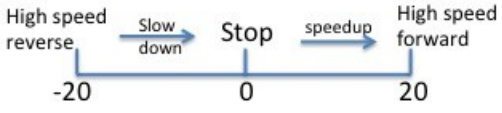
Release to Public

PWM Control Function

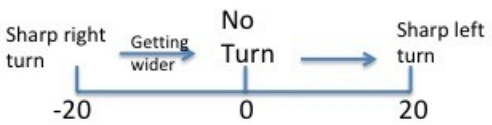
```
#define requireConstant     75
```

```
void AGV2000Drive(int speed, int turn){
    int speedResult=0, turnResult=0;
    speedResult = speed + requireConstant;
    turnResult = turn + requireConstant;
    unifiedContolBLDC(speedResult,turnResult,requireConstant);
}
```

Speed Range: [-20, 20]
-20 to 0: Reverse drive
0 to 20: Forward drive



Turn Range: [-20, 20]
-20 to 0: Right Turn
0 to 20: Left Turn



VII. Implement RTOS and TCP/IP client in NXP LPC1769

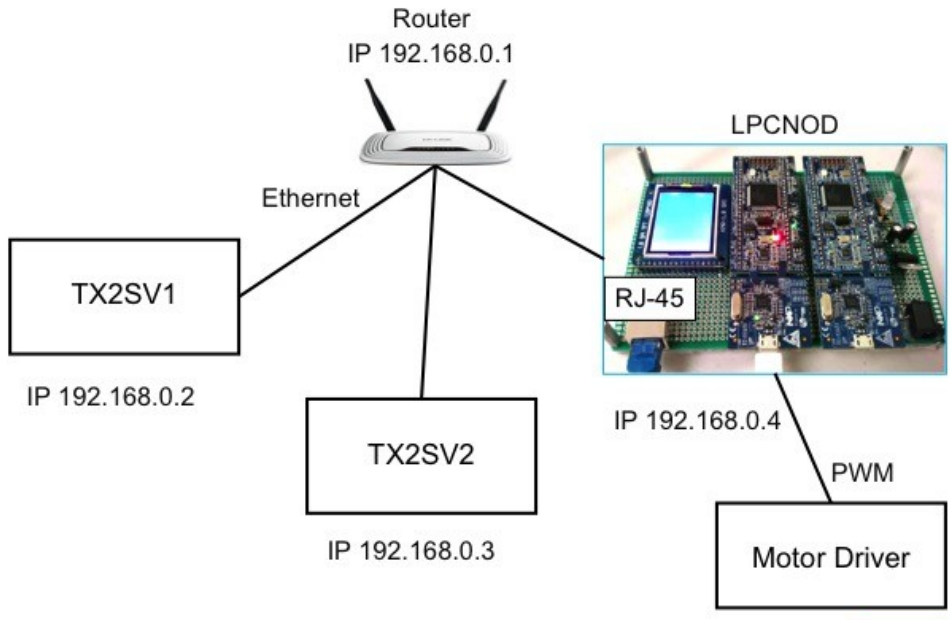
This lab counts total 10 points. Note hard copy of the lab report has to be ready for the time to make in class demo, fail to bring the hard copy can result in 5 marks reduction. In addition, the softcopy of the report including the source code exported as a project have to be submitted online. This lab is a preliminary step to use a microprocessor to control motor drive, you will

1. Understand Ethernet hardware and pin definition, and develop Ethernet hardware for LPC1769
2. Implement client server program and integrate with FreeRTOS to achieve multi-threading running on system
3. Use Linux system to give command to microprocessor through TCP/IP and achieve very low latency

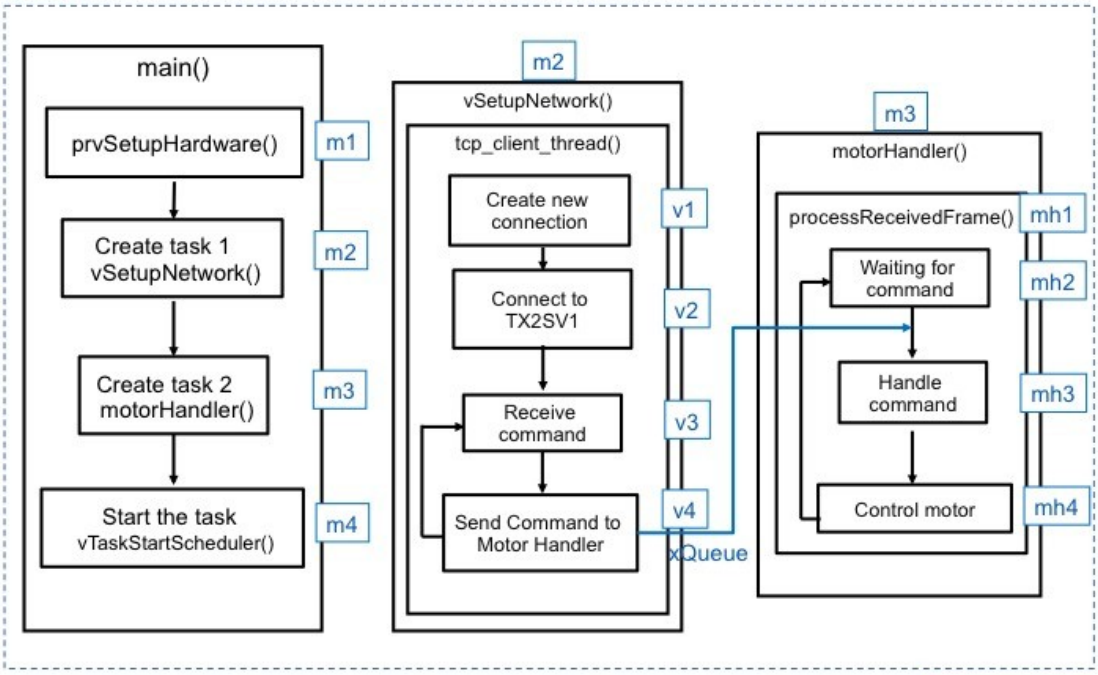
Project Introduction

- Objectives
  - Integrate LPC1769 with 100Mbps Ethernet to replace Modbus, achieved better scalability, lower round trip time
  - Implement FreeRTOS in the system to achieve multitasking, inter-task communication and better CPU utilization
- Progress
  - Implemented communication between TX2SVR1 and LPCNOD through TCP/IP
  - Completed testing on the vehicle integrated with TX2SVR1
  - Enhanced safety system: LPC will stop the vehicle if the TX2SVR1 disconnected

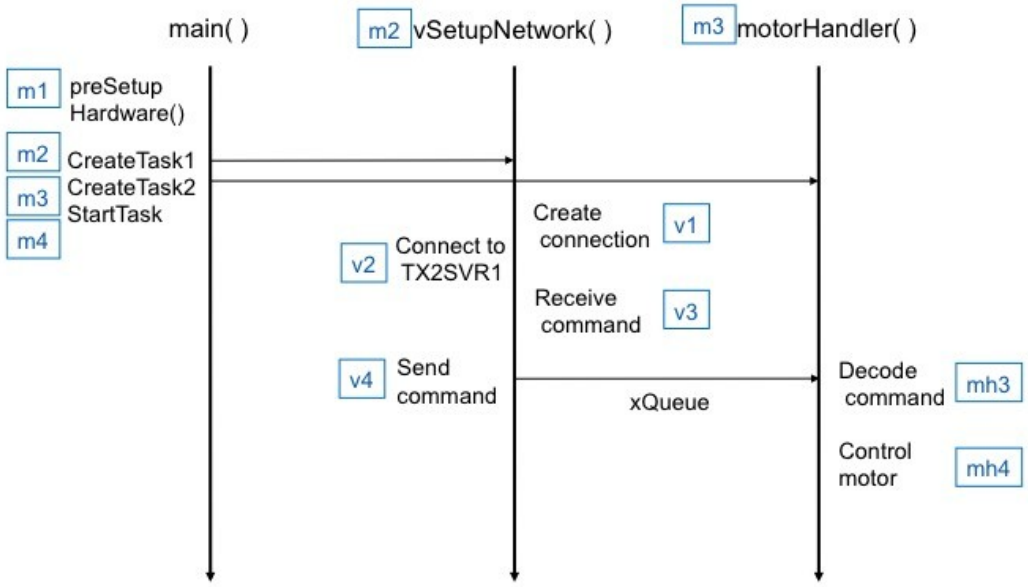
AGV4000 System Connection Diagram



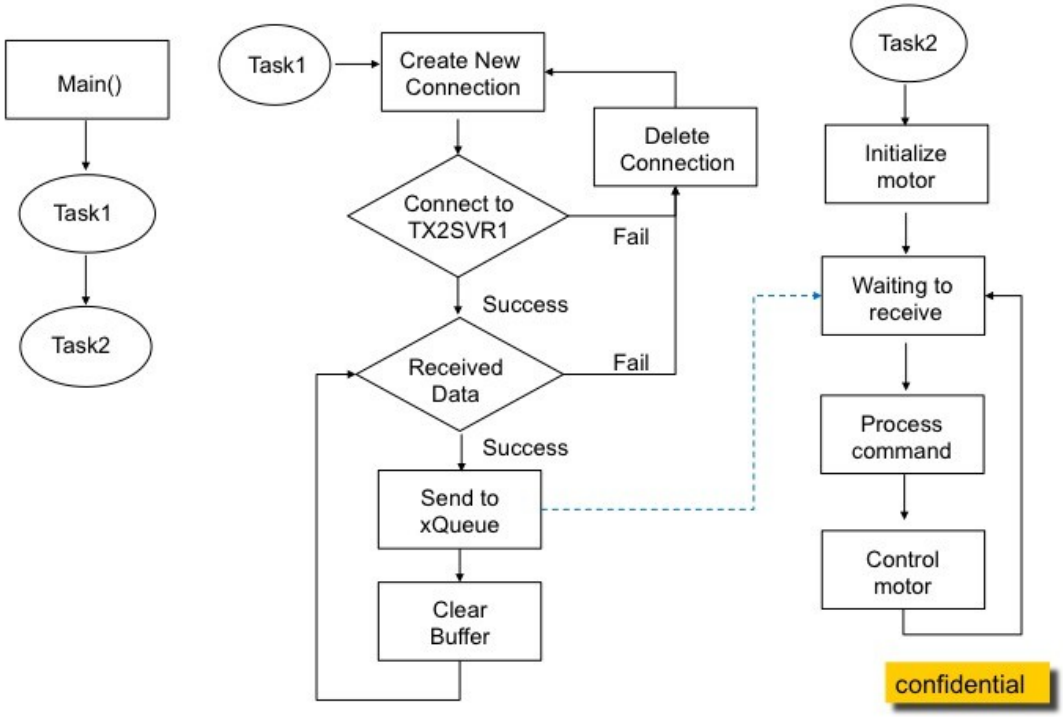
LPCNOD Program Overview



Communication Flow



Program Flow Chart



LPCNOD Main Program

```
int main(void)
    prvSetupHardware();                m1

    /* xQueue is for transferring data between threads. */
    xQueue = xQueueCreate( 1, 10 * sizeof( char ) +1);

    /* Thread for Ethernet interface and TCP connection. */
    xTaskCreate( );                    m2

    /* Add motor control thread to handle the data received */
    xTaskCreate( );                    m3

    /* Start the scheduler */
    vTaskStartScheduler();             m4
```

LPCNOD Motor Handler

```
xTaskCreate(motorHandler, (signed char *) "Motor_Handler",
                configMINIMAL_STACK_SIZE, &xQueue,
 (tskIDLE_PRIORITY + 1UL),  (xTaskHandle *) NULL);


void motorHandler(void *pvParameters) {
      xQueue = *(xQueueHandle *) pvParameters;

      for( ;; ) {
            processReceivedframe();          mh1
            vTaskDelay(1);//sleep for 1ms
      }
}

void motor_data_frame_init() {
      memset(&data_frame, 0, sizeof(data_frame));
      …
      char temp[10] = "";
      if(xQueueReceive(xQueue,  &temp, portMAX_DELAY)){   mh2
            …
}
```


LPCNOD Motor Handler

```
void processReceivedframe() {

      //    initialize the motor_data_frame
      motor_data_frame_init();                mh3
            …

      //get the control byte
      control_byte = data_frame.sign_action;

      switch (control_byte) {                 mh4
            …

      }
```

confidential

LPCNOD vSetupNetwork

```
static void vSetupNetwork(void *pvParameters) {
      …                                                          m3
      //Setup IP Address and
      netif_set_default(&lpc_netif);
      netif_set_up(&lpc_netif);

      //Start TCP client thread
      tcp_client_init();
      …
}

tcp_client_thread(void *arg){
      struct netconn *conn;…
      while(1){
            /* Create a new connection identifier. */
            conn = netconn_new(NETCONN_TCP);                     v1
            err = netconn_connect ( conn, &servIpaddr, TX2SVR1_PORT)  v2
            while ((err = netconn_recv(conn, &buf)) == ERR_OK) {  v3
                  …
                  xQueueOverwrite( xQueue, stopVehicle);         v4
            }
      …
      }
```

confidential

Test Result

Tested items:

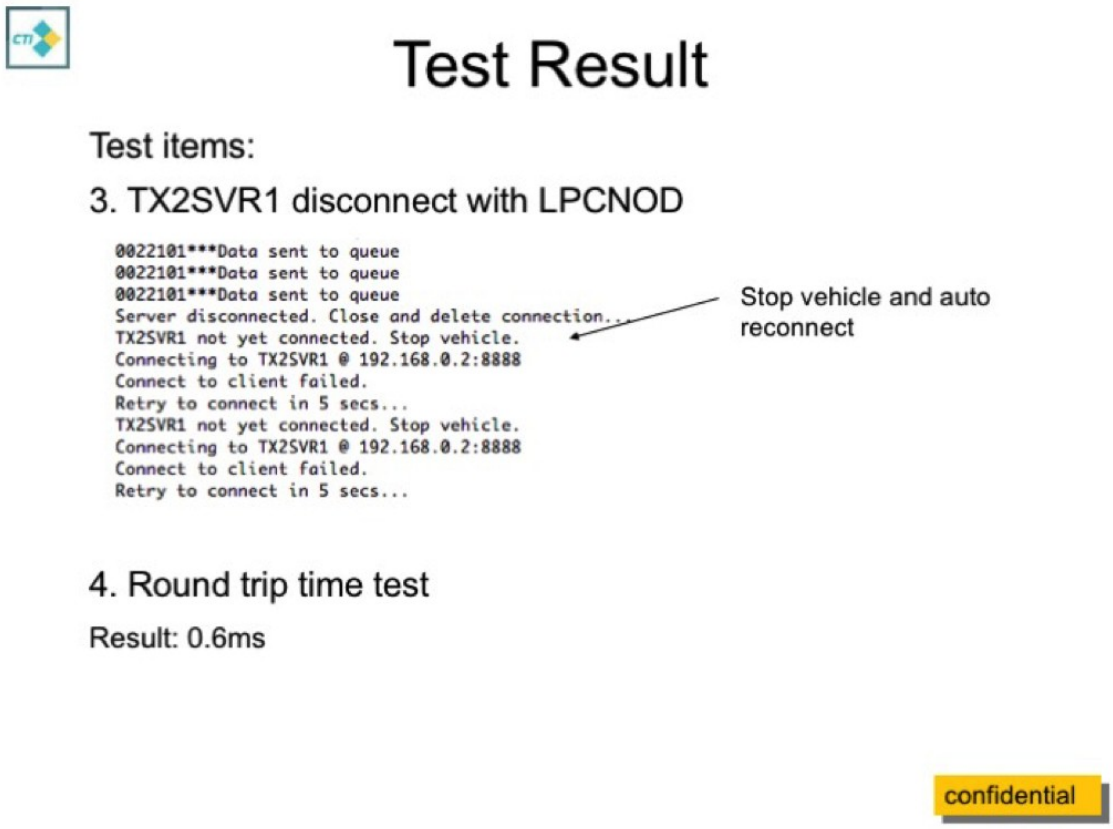
1.LPCNOD and TX2SVR1 connection test

```
Ethernet interface is initializing...
Waiting for TCPIP thread to initialize...
Starting LWIP TCP server...
LPC_IP_ADDR: 192.168.0.4        <-------- LPCNOD IP Address
NET_MASK   : 255.255.255.0
GATEWAY_IP : 192.168.0.1
Starting TCP client_thread ...
TX2SVR1 not yet connected. Stop vehicle.
Connecting to TX2SVR1 @ 192.168.0.2:8888   <-- TX2SVR1 IP Address
Connect to client failed.
Retry to connect in 5 secs...
```

2. LPCNOD command receiving test

```
Connecting to TX2SVR1 @ 192.168.0.2:8888
Connect to TX2SVR1 successful...
0022101***Data sent to queue
0022101***Data sent to queue
0022101***Data sent to queue
0022101***Data sent to queue
0022101***Data sent to queue
0022101***Data sent to queue
0022101***Data sent to queue
0022101***Data sent to queue
0022101***Data sent to queue
```

confidential

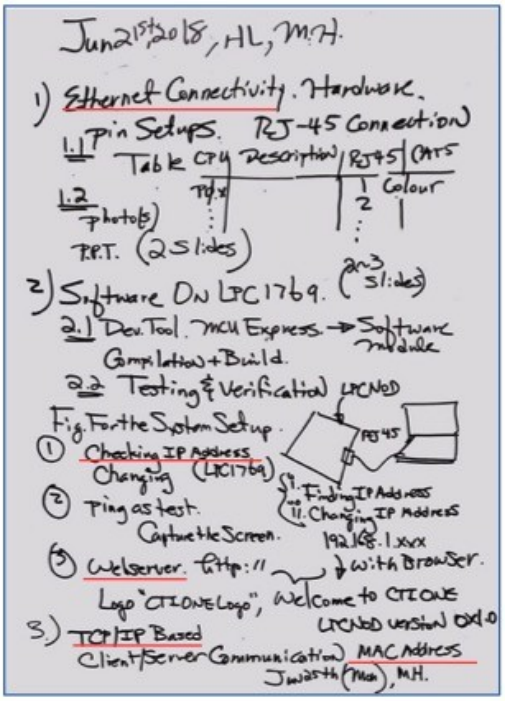


VIII. Implement Ethernet hardware and sample program in LPC1769

This lab counts total 10 points. Note hard copy of the lab report has to be ready for the time to make in class demo, fail to bring the hard copy can result in 5 marks reduction. In addition, the softcopy of the report including the source code exported as a project have to be submitted online. This lab is a preliminary step to use a microprocessor to control motor drive, you will

1. Design and implement Ethernet hardware and connect to LPC1769 board, and connect Ethernet cable to a router.
2. Implement a TCP program that can handle and respond to “ping”, and write a client server program to transmit and receive data through Ethernet
3. Understand RTOS function and architecture and know how to modify it.

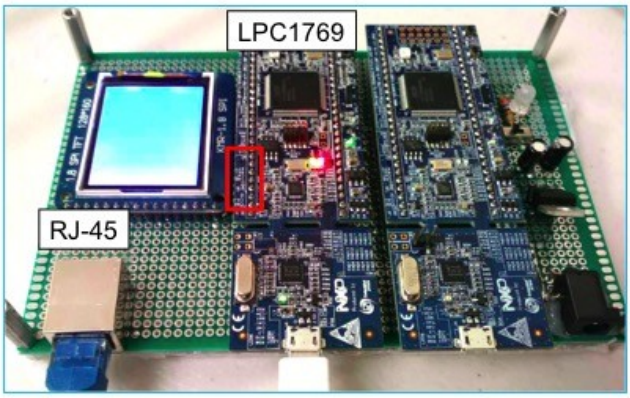
6-21-18 Ethernet Connection And Webserver



- Requirement and technical spec

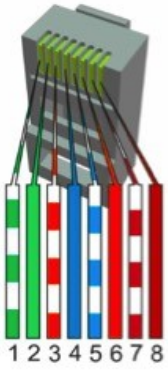
1. Ethernet hardware implementation
2. Client-server program development
3. Function test and verification

Ethernet Hardware Connection



RJ-45 Header

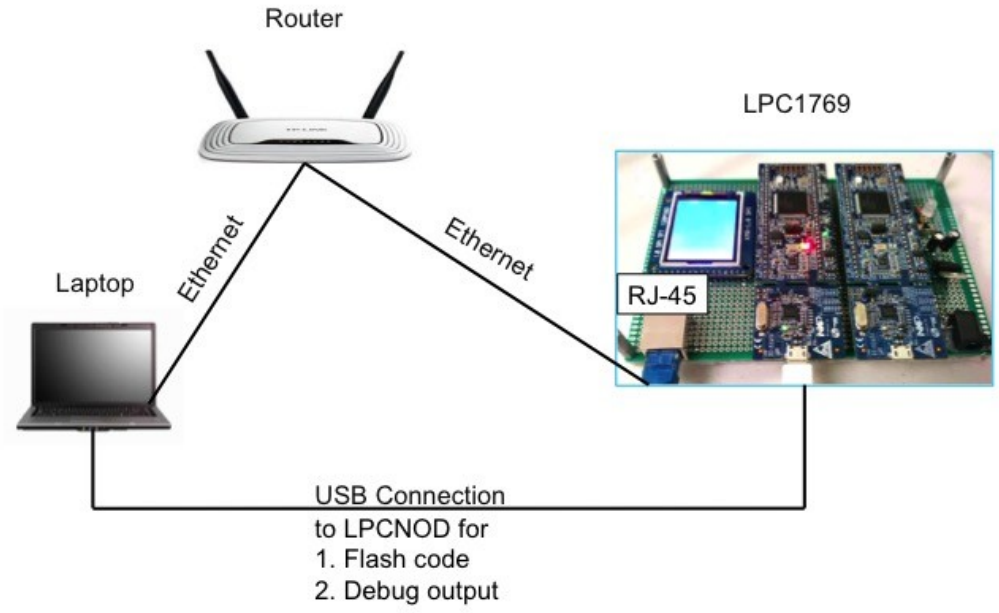
RJ-45 Cable



LPC1769 Module		RJ-45 Jack	
Pin	Definition	Pin	Definition
J2_32	ETH_RXN	6	RX-
J2_33	ETH_RXP	3	RX+
J2_34	ETH_TXN	2	TX-
J2_35	ETH_TXP	1	TX+

(2 differential pairs)

System Connection Diagram



Software Setup

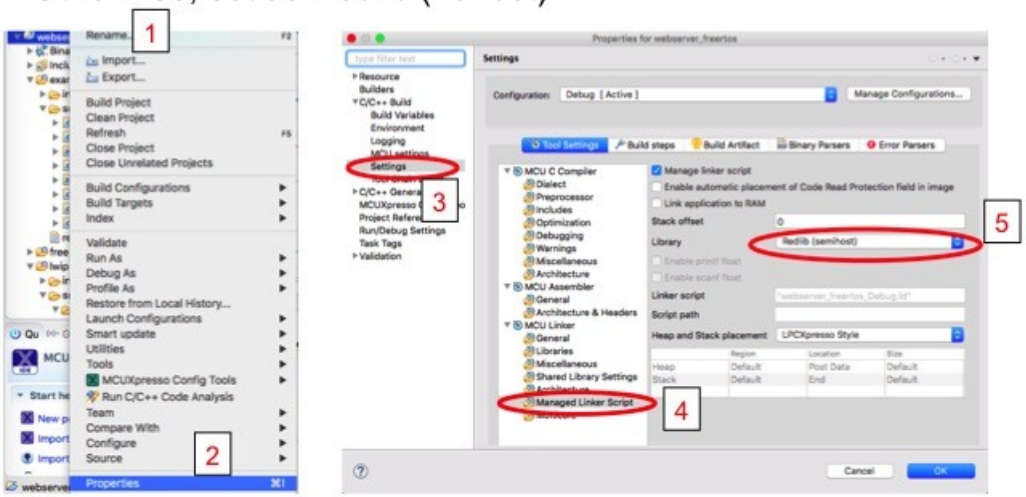
- Environment: MCUExpresso IDE v10.1.1 [Build 606] [Download]
- Test code: LPCOpen “webserver” example [Download]
- In “lwipopts.h”, config IP for DHCP or manual

```
114     /* Static IP assignment */
115 #if LWIP_DHCP
116     IP4_ADDR(&gw, 0, 0, 0, 0);
117     IP4_ADDR(&ipaddr, 0, 0, 0, 0);
118     IP4_ADDR(&netmask, 0, 0, 0, 0);
119 #else
120     IP4_ADDR(&gw, 192, 168, 0, 1);
121     IP4_ADDR(&ipaddr, 192, 168, 0, 3);
122     IP4_ADDR(&netmask, 255, 255, 255, 0);
123     printf(&ipaddr);
124 #endif
```

- LPC1769 IP setting: IP 192.168.0.3, GW 192.168.0.1, Mask 255.255.255.0
- PC IP setting: IP 192.168.0.2, GW 192.168.0.1, Mask 255.255.255.0
- Router IP setting: IP 192.168.0.1, GW 192.168.0.1, Mask 255.255.255.0

Software Setup for Debug

- To print debug information on MCUExpresso console
  1. Right click on project name -> properties
  2. Changed to Redlib (semihost)
- Otherwise, set as Redlib (nohost)



Testing and Verification

- PC: ping 192.168.0.3 successfully

```
Jerry — -bash — 82×23
JerrysMacBookAir:~ Jerry$ ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: icmp_seq=0 ttl=255 time=0.269 ms
64 bytes from 192.168.0.3: icmp_seq=1 ttl=255 time=0.306 ms
64 bytes from 192.168.0.3: icmp_seq=2 ttl=255 time=0.272 ms
64 bytes from 192.168.0.3: icmp_seq=3 ttl=255 time=0.319 ms
64 bytes from 192.168.0.3: icmp_seq=4 ttl=255 time=0.361 ms
64 bytes from 192.168.0.3: icmp_seq=5 ttl=255 time=0.302 ms
64 bytes from 192.168.0.3: icmp_seq=6 ttl=255 time=0.297 ms
^C
--- 192.168.0.3 ping statistics ---
7 packets transmitted, 7 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.269/0.304/0.361/0.029 ms
```

- Use Netcat (NC) to test the IP and port:

```
JerrysMacBookAir:~ Jerry$ nc -vvv 192.168.0.3 6001
found 0 associations
found 1 connections:
     1: flags=82<CONNECTED,PREFERRED>
        outif en3
        src 192.168.0.1 port 52235
        dst 192.168.0.3 port 6001
        rank info not available
        TCP aux info available

Connection to 192.168.0.3 port 6001 [tcp/*] succeeded!
```

Client-server Testing

- Message loop-back test using Netcat:
  - Type message and enter, and LPC will send back

NC Console Output

```
Connection to 192.168.0.3 port 6001 [tcp/*] succeeded!
Hello! You have connected to LPC1769!
Please type in message:
Hi!!
Hi!!
This is CTI One Developer Jerry.
This is CTI One Developer Jerry.
Everything is good.
Everything is good.
```

LPC Console Output

```
webserver_freertos LinkServer Debug [C/C++ (NXP Semiconductors)]
IP_ADDR    : 192.168.0.3
NET_MASK   : 255.255.255.0
GATEWAY_IP : 192.168.0.1
Starting server_thread ...
Waiting connection...
IP_ADDR    : 192.168.0.3
NET_MASK   : 255.255.255.0
GATEWAY_IP : 192.168.0.1
Accepted a new connection 0x10003f8c
[PC_NOD] Hi!!
[PC_NOD] This is CTI One Developer Jerry.
[PC_NOD] Everything is good.
```

  - Transmit Round trip time (1kB and 2kB)

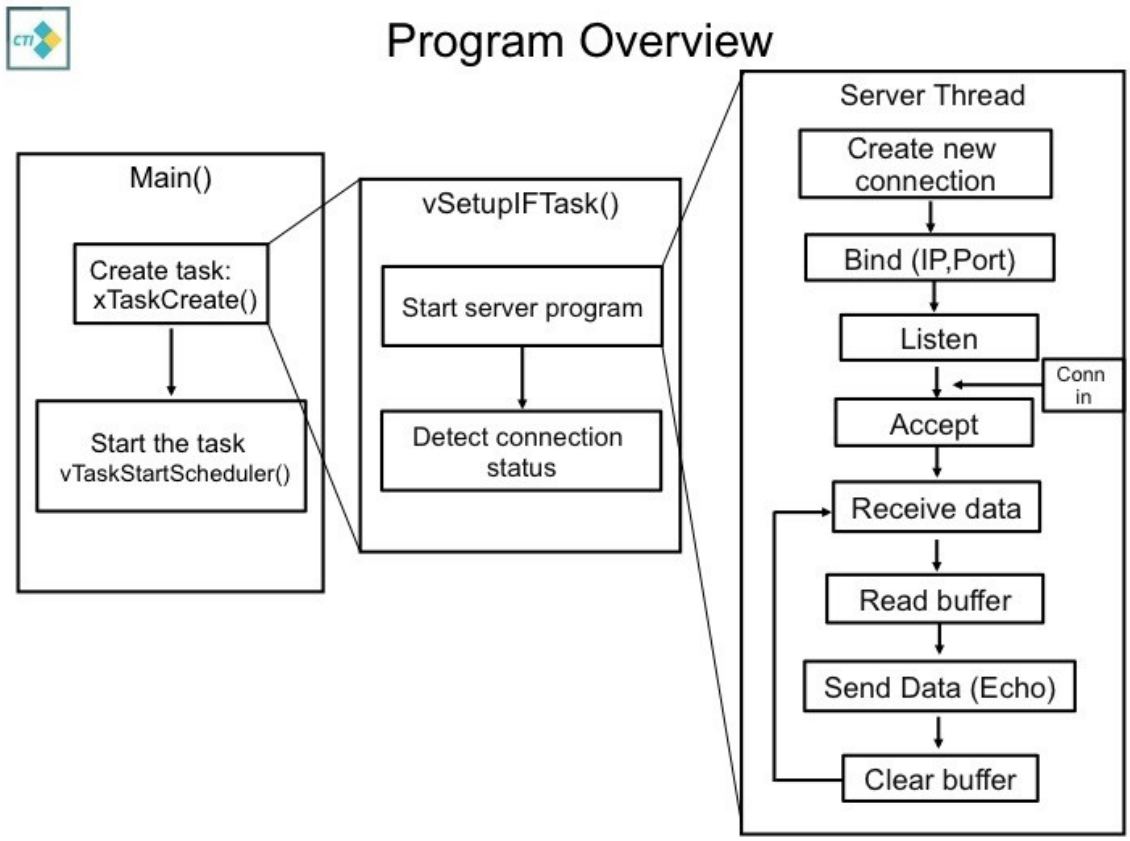
```
JerrysMacBookAir:~ Jerry$ ping 192.168.0.3 -s 1024
PING 192.168.0.3 (192.168.0.3): 1024 data bytes
1032 bytes from 192.168.0.3: icmp_seq=0 ttl=255 time=0.924 ms
1032 bytes from 192.168.0.3: icmp_seq=1 ttl=255 time=0.949 ms
1032 bytes from 192.168.0.3: icmp_seq=2 ttl=255 time=0.992 ms
1032 bytes from 192.168.0.3: icmp_seq=3 ttl=255 time=0.904 ms
1032 bytes from 192.168.0.3: icmp_seq=4 ttl=255 time=0.943 ms
^C
--- 192.168.0.3 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.904/0.942/0.992/0.029 ms
```

```
JerrysMacBookAir:~ Jerry$ ping 192.168.0.3 -s 2048
PING 192.168.0.3 (192.168.0.3): 2048 data bytes
2056 bytes from 192.168.0.3: icmp_seq=0 ttl=255 time=1.573 ms
2056 bytes from 192.168.0.3: icmp_seq=1 ttl=255 time=1.564 ms
2056 bytes from 192.168.0.3: icmp_seq=2 ttl=255 time=1.634 ms
2056 bytes from 192.168.0.3: icmp_seq=3 ttl=255 time=1.545 ms
2056 bytes from 192.168.0.3: icmp_seq=4 ttl=255 time=1.545 ms
^C
--- 192.168.0.3 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.545/1.572/1.634/0.033 ms
```

HTTP Server Testing

- Message loop-back test using Netcat:
  - Type message and enter, and LPC will send back

LPC Console Output



IX. Magnetic Sensor implementation with LPC1769.

This lab counts total 10 points. Note hard copy of the lab report has to be ready for the time to make in class demo, fail to bring the hard copy can result in 5 marks reduction. In addition, the softcopy of the report including the source code exported as a project have to be submitted online. This lab is a preliminary step to use a microprocessor to receive multiple analog inputs from the magnetic sensor MGS1600GY:

lec9-5-MagSEN_LPC-HL-PR-2018-9-12.ppt

Magnetic Sensors to LPC1769 Connection

CTI One Corporation

Version: x0.1

Date: September 2018

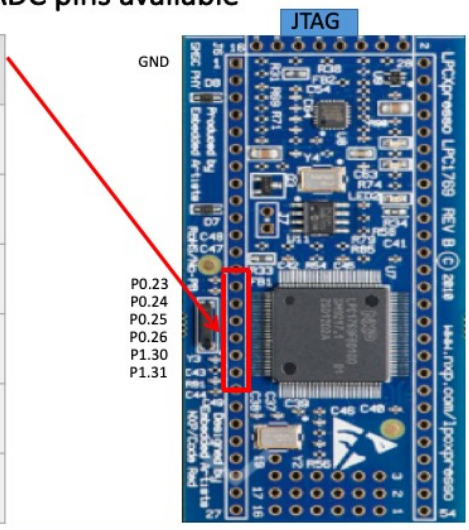
Project Lead: Harry Li, Ph.D.

Team members: Prashanth Rajasekar

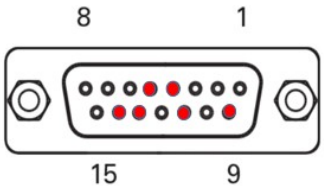
LPC1769 ADC pin Layout

6 ADC pins available

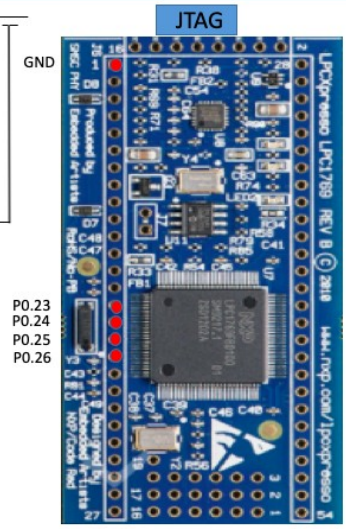
Available ADC pins	
AD0	P0.23
AD1	P0.24
AD2	P0.25
AD3	P0.26
AD4	P1.30
AD5	P1.31



Magnetic Sensor 1 Pin Connectivity Table

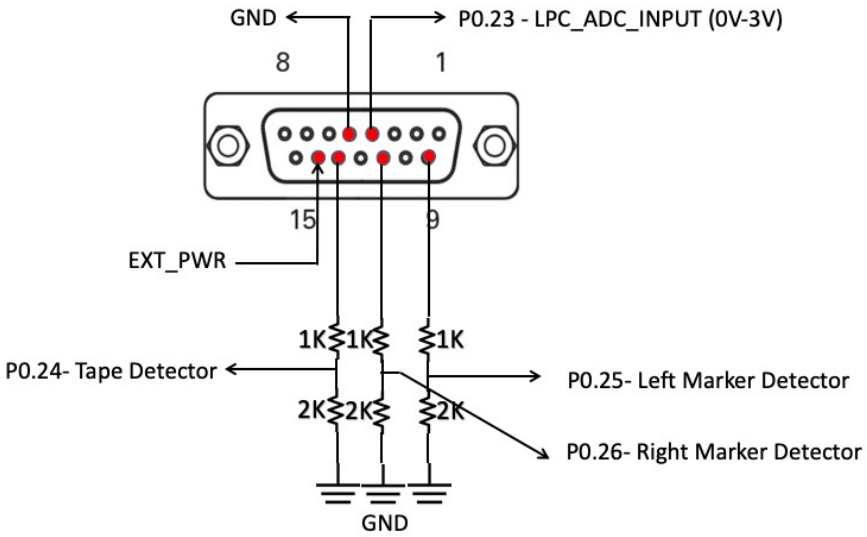


MagSen1 Pins	LPC1769 Pins
Pin 5	GND
Pin 14	EXT_PWR
Pin 4	P0.23
Pin 13	P0.24
Pin 9	P0.25
Pin 11	P0.26

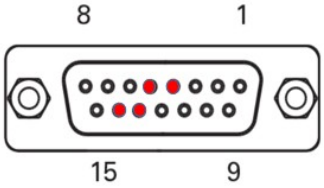


Input Data from MagSen1： TrackDetected， Voltage 0V-3V， Left_Marker_detection, Right_Marker_detection

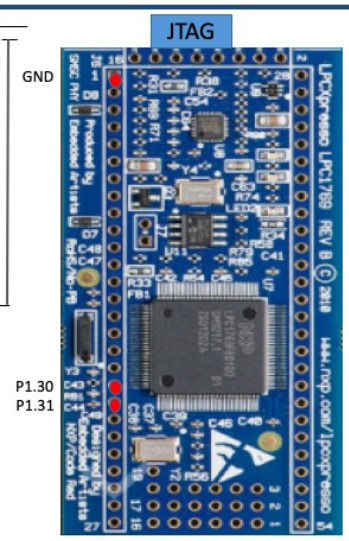
Magnetic Sensor 1 Physical connection



Magnetic Sensor 2 Pin Connectivity Table

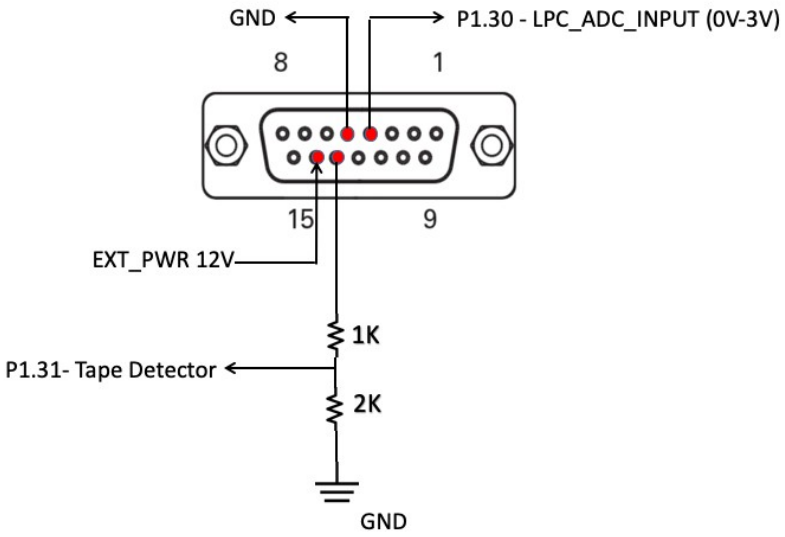


MagSen1 Pins	LPC1769 Pins
Pin 5	GND
Pin 14	EXT_PWR
Pin 4	P1.30
Pin 13	P1.31



Input Data from MagSen2： TrackDetected，Voltage 1V-3V

Magnetic Sensor 2 Physical connection



Enable GPIO pin as below.

```
int main(void) { ////////////////////// Main program ////////////////////////////////

#if TESTING_PWM
#define RANDOM              0x2222
#endif

    unifiedMotorInit();
    //Chip_GPIO_WriteDirBit(LPC_GPIO, 0, 24, false);
    LPC_GPIO0->FIODIR &= !(0x1 << 24);
    LPC_GPIO0->FIODIR &= !(0x1 << 25);
    LPC_GPIO0->FIODIR &= !(0x1 << 26);
    LPC_GPIO1->FIODIR &= !(0x1 << 31);
```

Fetch the data as below.

```
if(LPC_GPIO0->FIOPIN >> 25 & 1){////////////// Left marker
    //  AGVStraightForward(8, 8);

else if(LPC_GPIO0->FIOPIN >> 26 & 1){ //right marker
    marker_flag+=1;
    if(marker_flag%2==0){

if(LPC_GPIO0->FIOPIN >> 24 & 1 ){
    InitADC(MSen.adcChannel);
    //printf("S2\n");
    /*Start ADC conversion*/

if(LPC_GPIO1->FIOPIN >> 31 & 1 ){//Chip_GPIO_ReadPortBit(LPC_GPIO, 0, 24)){
    /*Start ADC conversion*/
    InitADC(MSen2.adcChannel);
    //printf("yes\n");
    LPC_ADC->ADCR |= (1<<START_BIT);
```

Appendix A. 6 Projects Requirements

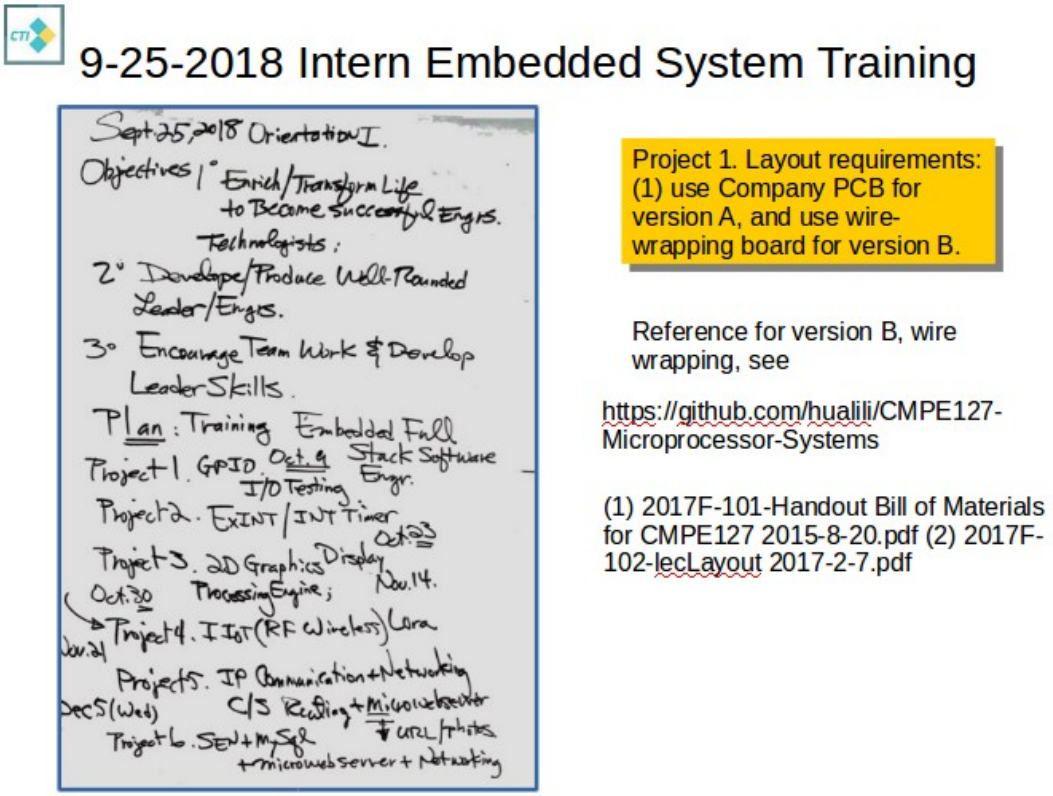


Figure A.1. Six projects for embedded software engineer training.

Appendix B. IEEE Style Report Template

Author Guidelines for Project Report Writing

Hua Harry Li, Ph.D.
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: hualili@yahoo.com

Abstract

An abstract is one of the most important parts of the report. It should be written in a brief and factual manner with no more than 50 words, and very often one paragraph. It should give one sentence clear description of project, then the goal and objective(s), the motivation and/or the significance of the project. Then it should give one sentence of the main technical challenge(s), and how did you overcome this challenge/difficulty. Finally, one sentence gives the final result, most importantly indicate if the goal was accomplished.

1. Introduction

One or two paragraph gives the background information (provide references if needed, and list these reference in the Reference Section [1]) and provides further detailed information of the project goal and objective(s), technical challenges and /or difficulties, and brief description of how to approach and solve this challenge/difficulties.

2. Methodology

First give one or two sentences to point out what you want to provide to the reader in this section and how these material is organized. Then very often you want to cover the following subjects in this section.

2.1. Objectives and Technical Challenges

Give the detailed, very often are enumerated objectives which can be derived from the goal of the project, then describe briefly the corresponding technical challenges.

2.2. Problem Formulation and Design

Give the detailed, one-to-one correspondence description of your design to attach/solve the problem and to achieve the objectives and the goal of the project:

1. Use engineering language and mathematical formulation;
2. Provide system block diagrams and circuit schematics for your hardware design;
3. Give flow chart and pseudo code description for the step-by-step discussion of your software design.

3. Implementation

One sentence to describe the organization of this section, then provide detailed discussion on your implementation.

3.1. Hardware Design

Provide the following description and discussion:

1. System block diagram(s).
2. Schematic design.
3. Bill of material (list of components).

3.2. Software Design

Provide the following description and discussion:

1. Flow chart or flow charts, very often you should provide one top level flow chart, then additional flow charts for detailed lower level implementations.
2. Algorithm, e.g., description of the step by step implementation.
3. Pseudo code for each section of the implementation.

4. Testing and Verification

Provide testing and verification procedures to make sure quality control personnel, team members, supervisor can verify and reproduce the experimental result.

5. Conclusion

Provide summary of this project, briefly review the statements made in the abstract, in particular, if the enumerated objectives and goal are achieved. Emphasize and highlight the lessons learned, point out the direction for further improvement if needed.

6. Acknowledgement

Provide acknowledgement if needed, such as support, help, or assistance from someone. These support, help, assistance are crucial.

7. References

[1] H. Li, "Author Guidelines for CMPE 146/242 Project Report", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

8. Appendix

Diagrams, source code listing, circuit schematics, relevant datasheets etc. go here.

(End)