Embedded Software

CmpE244

Sept. 29 (Wed) 4:00-8:00pm.

Zoom Link To Be used for the Entire Semester

Harry Li, Office: Eng. Building Rm 269A

E-mail: hua.li@sjsu.edu.

Text messages (650)400-6166.

Grading Policy:

1° Projects & Assignment 30%

2 mandatory Projects, 10%×2 = 20%

1 Semester-Long Project 10%

2° Midterm: 30%

3° Final: 40%

Organization of the Course

1. CPU Architecture, memory map. Special Purpose Registers for the init & config of Peripheral Controller. Firmware Development. (~3 weeks)

2. Kernel (O.S) Source Distribution IDE (Integrated Development Environment), To be able to optimize Kernel image, to be able to modify existing Device Drivers. to write your own Device Driver. (~3 weeks)

3. Integration & Development of O.S. Kernel + Device Driver + Sensors/Actuators

Stepper Motor Drive

Sensors LSM303

P.I.D controller.

Fourier Transform.

Web Server (GUI)

OpenCV, OpenGL

Introduction

1. Development Setup

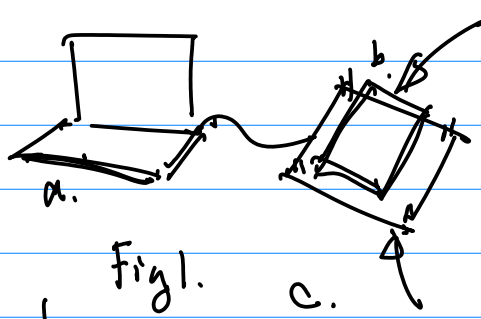


Fig 1.

c. Wire Wrapping Board

a. Host PC/Laptop, Linux Ubuntu 18.04

(Virtual Box Installed, then install Linux on top of it.

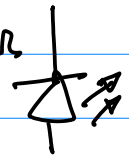
b. Target platform (To Be determined)

c. Wire Wrapping Board

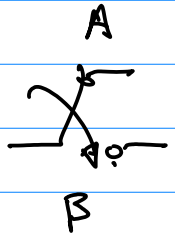
① ~$3\frac{1}{2}$" x 4" physical Dimension Through Holes with metal coating.

(2) 4 mounting Holes @ the corners. 4 stand-offs (Legs)

(3) LED Red/Green Current $\leq 10mA$
Resistors $V_{LED} \cong 1.8 VDC$
200 ~ 500Ω



(4) Toggle switch



3. Selection & Evaluation of A Target platform

Linux O.S. Support

ARM (RISC — Reduced Instruction Set Computer)

- Most Efficient Computation Density Per Unit Power
- Reduced Instruction Set (Smaller collection of machine code instructions → Better Optimization of compiler)
- Forward Looking — Longer Life Cycle

2. O.S. Architecture

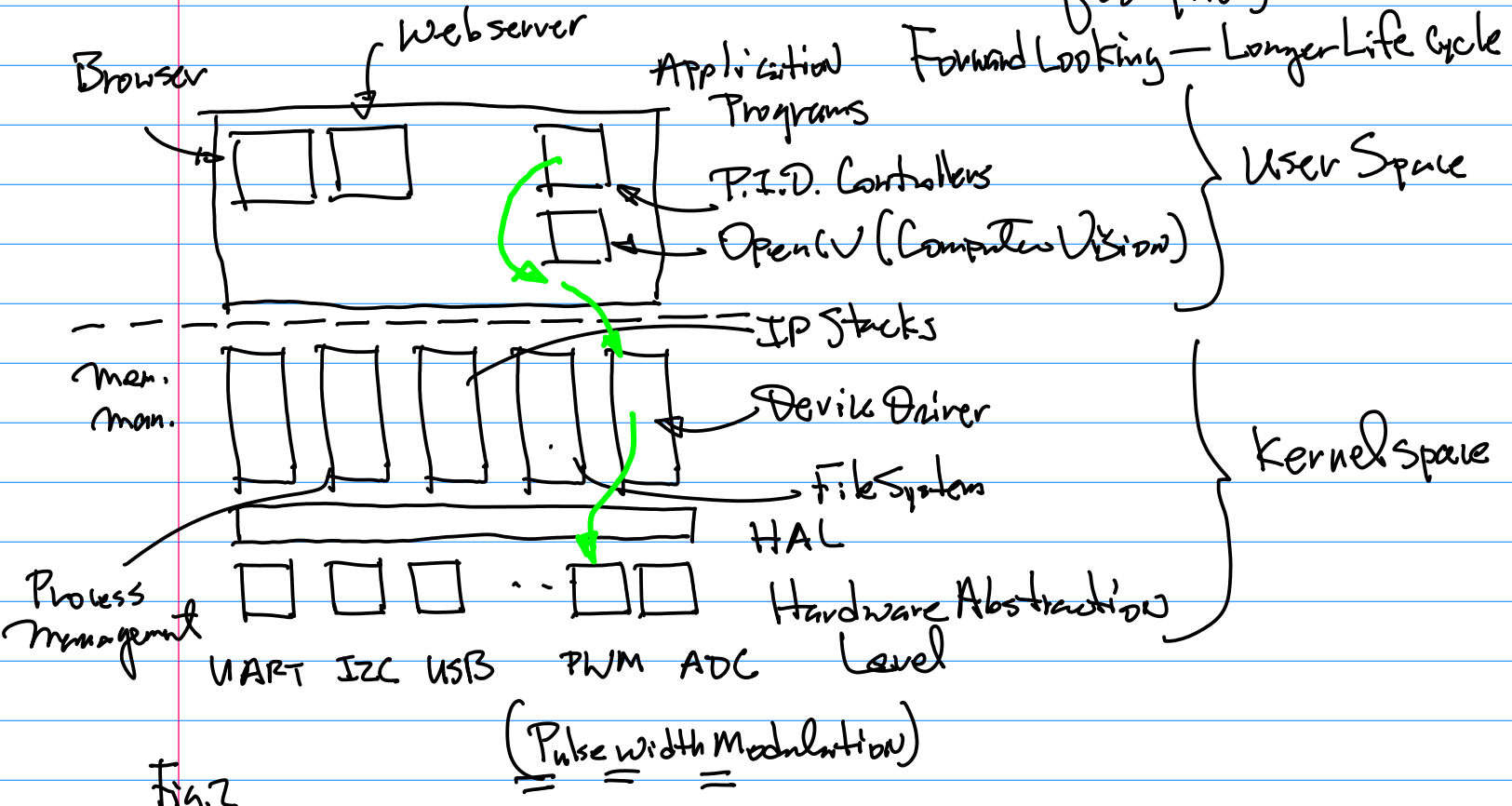


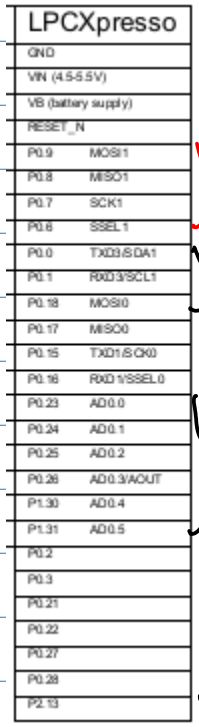
Fig. 2

Target Platforms To consider

1. NXP LPC17xx, 1769
   Clock Rate: 72MHz – 100MHz
   RTOS But Not Unix OS.
   No Linux
   Rich I/O Interface
   Ref: git~2021F-107b- . .

CPU Datasheet —— CPU Architecture well Documented
Memory map well Documented

Example: Sch of LPC1769.



"3+1" pin: SPI1 (Serial Peripheral Interface)

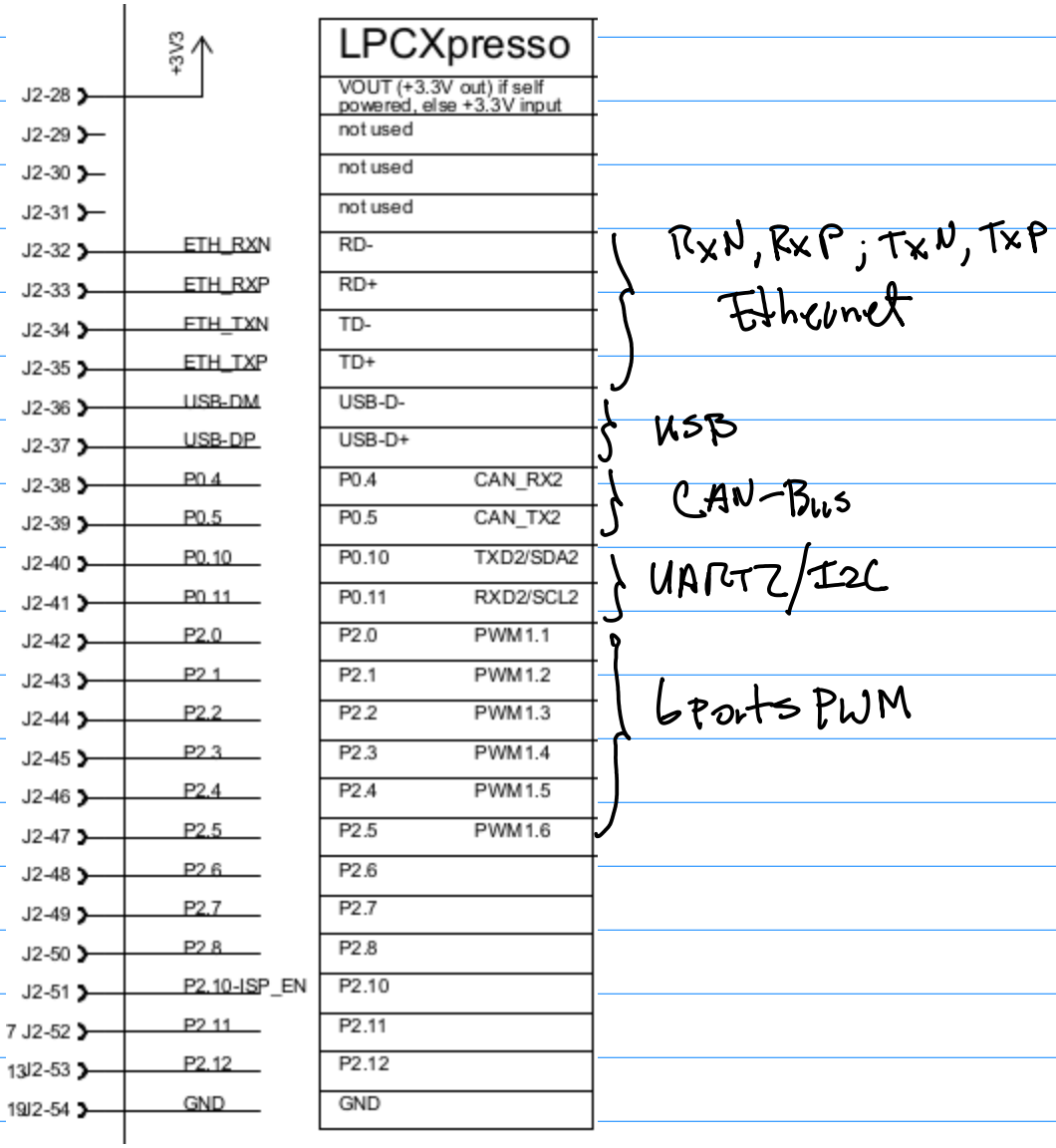
UART3 { Tx Transmission / Rx Receiving } multiplexed

SPI0/UART1

SDA: Serial Data
SCL: Serial CLK
I2C

6 ADC (AD — Analog to Digital Conversion)

GPP (GPIO: General Purpose Port I/O)



Option Target platform: FPGA. Futus Electronics.

FPGA Igloo2: RISC-V.

Superset of ARM Architecture

IP Core: Open Source. Supports RTOS

Limitations: No Unix/Linux O.S.

Smaller Gate Counts. → less

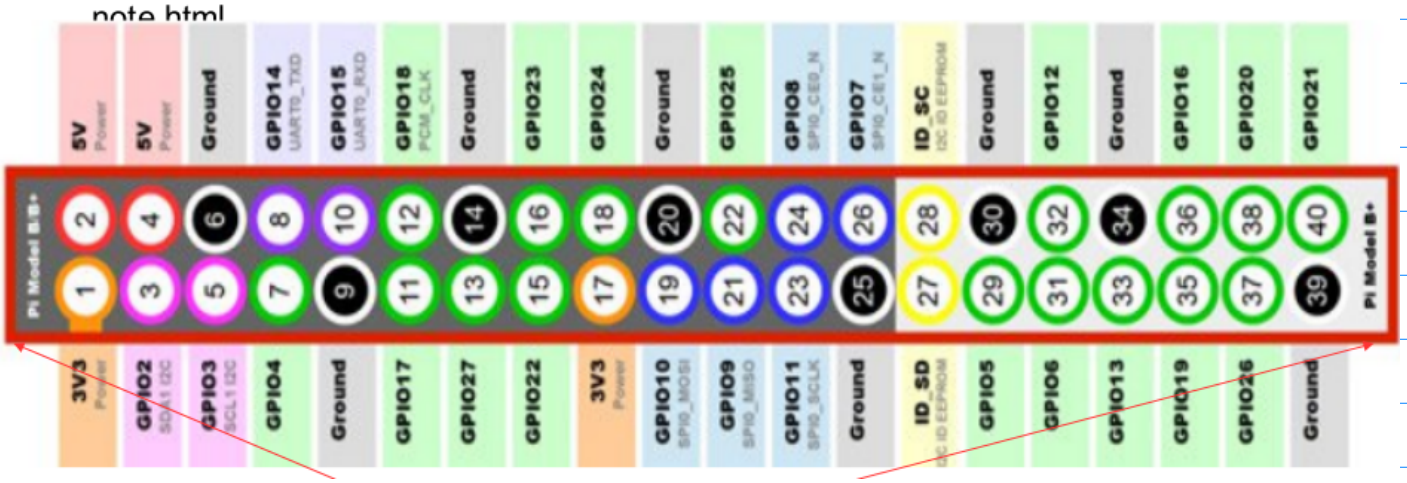
Computational Capability.

Pie. Broadcom BOM

Features: Support Linux/Ubuntu.
Provides machine/Computer Vision Capability; OpenCV.
YoLo (You only Look Once) — Deep Learning, NO ADC

Pie-3 Version B GPIO Pins

https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html



Plus: CPU Datasheet → Device Driver Development.
CPU Architecture (memory map, Peripheral Controllers, G.E. (Graphics Engine)

Option: ARM-11 Samsung. CPU: S3C6410x
CPU Datasheet: Architecture Block Diagram — well Document
600~800MHz memory map, well Designed, documented
Support Linux, well Document/Sample code for Driver Development.

State-of-the-Art Feature: Graphics Processing Engine — GPU
LPC17xx, NO; FPGA RISC-V, NO; BCM-Pie G.E. yes
ARM-11. Video Codec, Marginal

Option: NVDA — Jetson TX2 6 CPUs + 256 GPU in a Single Package

Supports Linux/Ubuntu, I/O Interface is limited ; (LPC17xx has the Best I/O I/F Support) ; ~$70 Dev. Kit. Well Documented Datasheet.

Option: NVDA Jetson NANO. Ubuntu Linux Support.
Multiple Cpus + 128 GPU
{ I/O (Limited)
{ Datasheet — Not As Detailed as other platform)

Developer forum is very Active and it gives a good references.

Homework: 1° Form 2-4 person team
By Next week ;
2° Choose Target Platform
3° Bring WireWrapping Board/ Prototype Board to the Class for show & Tell ;
4° Sign up @ Nvidia website as a developer → Kernel (Ubuntu)
{ Source Distribution
{ Jetpack

Nxp, MCU Xpresso
www.nxp.com

Sign up as a developer at nxp website
www.nxp.com
MCU Xpresso

Oct 6 (Thursday).
Topics:
1° Architectural Aspects of Embedded System for Software Implementation
Datasheet + Board Sch. + Special Purpose Register + IDE (Compiler and Flash Tool)

Example: LPC1769 ARM Cortex M3
CPU Datasheet
Simpler Architecture
(1) NXP LPC 1769 Baseline
(2) SamSung ARM11 Datasheet
CPU Block Diagram
memory map
Special Purpose Register
Tool Chain / Software Design, Implementation

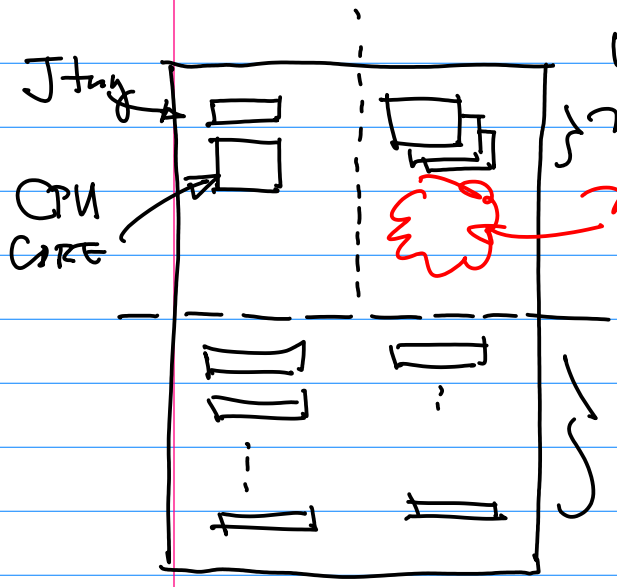


Fig1.

Note: 1. Enumeration of Subsystems is always started as 0.

Vector Graphics Engine { 2D Graphics { a. Display Device Driver, b. 2D primitive Graphics Processing, c. 2D Transformation { line, circle } } 3D Graphics

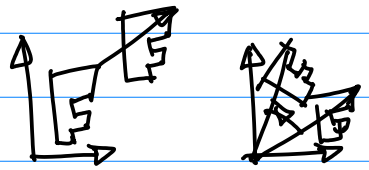


Fig.2b

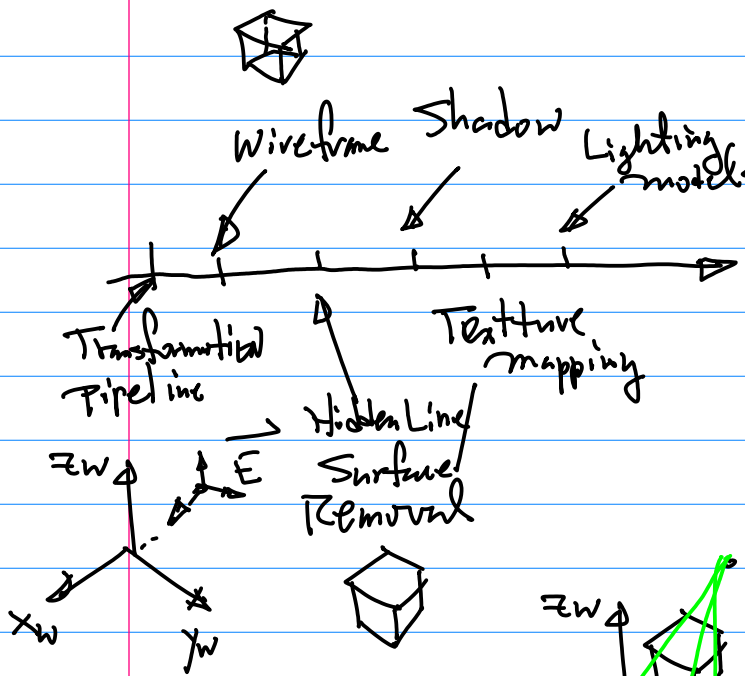
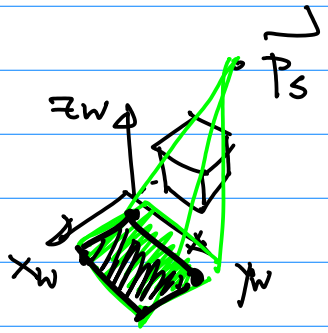


Fig.2



Memory map:

1. RISC: Reduced Instruction Set Architecture

1979 David Patterson
1982 John Hennessy

{ Uniformity
Regularity
Orthogonality

2. 32 Bit RISC Architecture

Data Bus: 32 bit, Bi-directional

D[m:n] Vector Notation

most Significant Bit — Least Significant Bit

D[31:0]

Endian "Little Endian"

Address Bus: 32 bit, Uni-Directional

ALU (Arithmetic/Logic Unit) 32 bits

Register File: 32 bit

{ General Purpose Registers: 32 bit
GPRs
Special Purpose Registers: 32 bit
SPRs

GPRs: those that can participate
Any meaningful Arithmetic
Logic Operations.

SPRs: Those Registers to fulfill special functions, such as init & Config. for Peripheral Controllers.

3. Memory map

a. Byte Addressable machine

The Smallest memory cell with an unique Address is a Single Byte — "Byte Addressable machine".

Question: What is the max Address for the memory map of a given CPU?

Single 32 Bit Architecture → Addr. Bus 32 bits

$2^{32} = 2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 2^{2}$

2^{10} ... 1K

2^{20} ... 1K x 1K = 1 Meg.

2^{30} ... 1M x 1K = 1 G.

4 GByte (Byte Addressable)

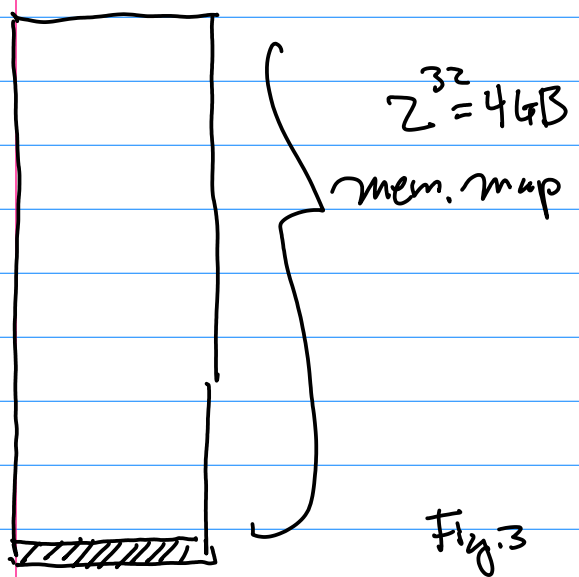


Fig.3

0X0000_0000 System Power_Up Address:

The Address when CPU is powered up, it will go to this memory location to fetch the 1st instruction to execute.

BANKS: A Block of memory.

Divid memory Map into 8 Equal Banks.

1st Bank: BANK 0
2nd Bank: BANK 1
⋮
8th BANK: BANK 7.

Question: How many Address Bits Do we need to define each memory bank? 3 bits

Question: Which 3 bits?

Addr. [31:0]

$a_{31}a_{30}a_{29} \cdots a_2a_1a_0$

Addr[31:29] = $a_{31}a_{30}a_{29}$

Question: Find the Starting Address of Each memory Bank?

Sol:

$a_{31}a_{30}a_{29}$	a_{28}		
0 0 0	0	0X0000_0000	BANK0
0 0 1	0	0X2000_0000	Bank1
0 1 0	0	0X4000_0000	Bank2
⋮			⋮
1 1 1			Bank7

Datasheet, pp.14 from NXP LPC1769 memory map, SSP0 as an example. then, starting Addr. for memory Banks,

Starting Address SSP0:

0X4008_8000

Question: How much memory does SSP0 need?

Peripheral Controllers

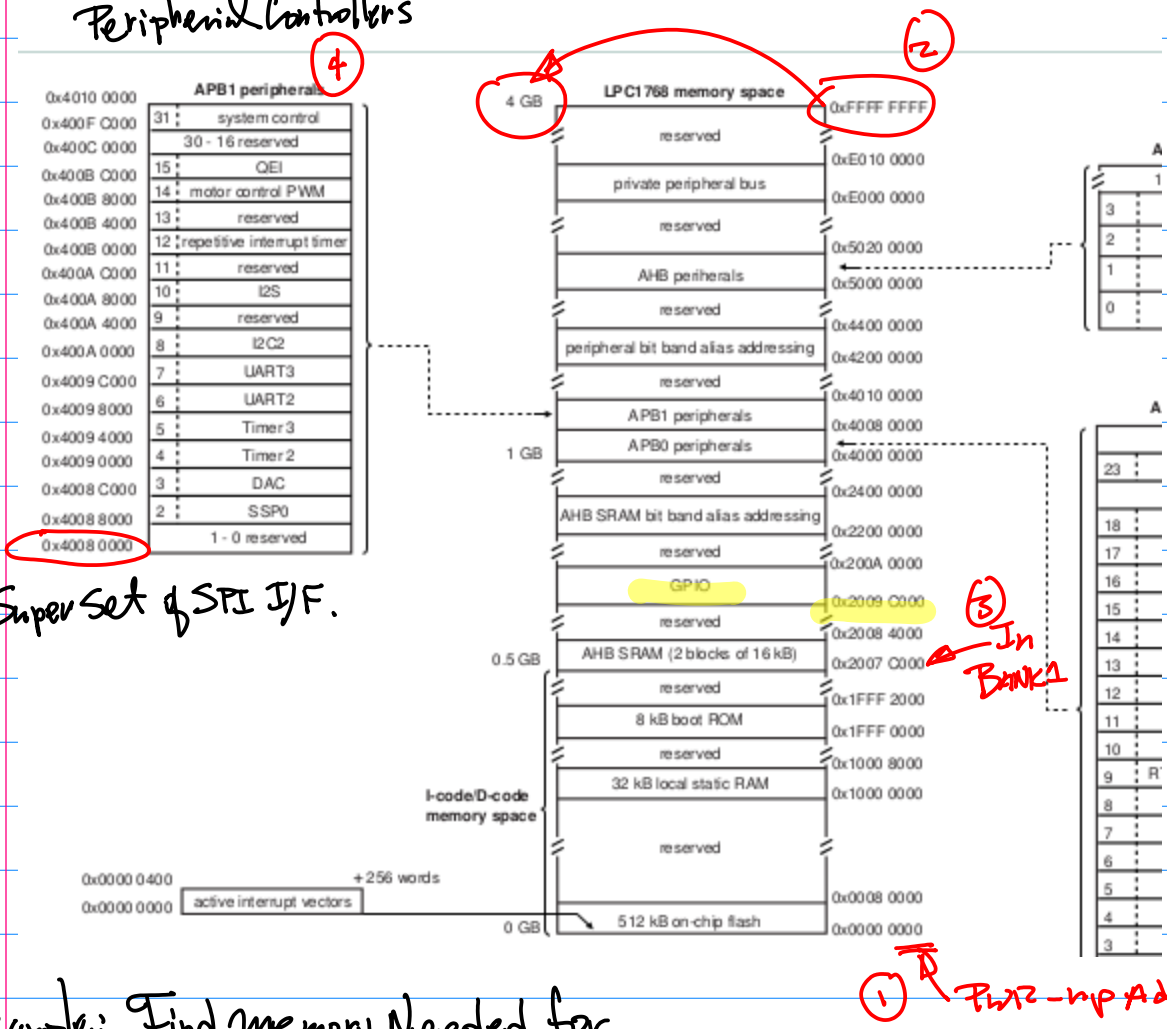


Fig 4

SSP0: Superset of SPI I/F.

Example: Find memory Needed for SSP0.

Starting Addr: 0X4008_8000

End Addr. 0X4008_C000 - 1

= 0X4008_bFFF

8000_bfff Block of memory

For what purpose? Employed By Special Purpose Registers for init & configuration of SSP0,

And to perform Data Input/Output Operation

Special Purpose Registers Design.

focus on GPP (General Purpose Port, eg. GPIO)

3 Common Types of SPR

Control Register : Init & Config.
Data Register : Data I/O Operation
Pull up/Down : Electrical Characteristic of the Controller

4. Naming Convention of Special Purpose Registers.
Let's Design/Define Naming Convention.
follow RISC Design Guidelines

Prefix + Root + Postscript

3 Letters 3 Letters 3 letters

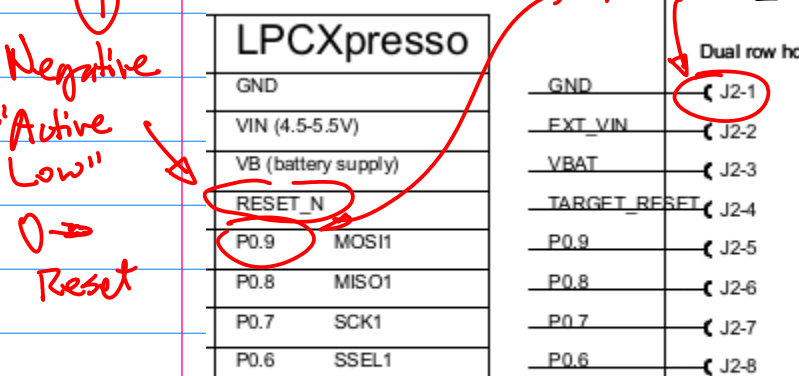
Control Register : GPx CON
GPx : General Purpose Port x
meaning we have more than one general purpose Ports.

C. Typical Number of GPP :
LPC1768 P0, P1, P2, P3.
Samsung Arm 11 :
17 GPPs

S3C6410 includes 187multi-functional input/o

PortName	Number of Pins.	
GPA port	8	UAI
GPB port	7	UAI
GPC port	8	SPI
GPD port	5	PCI
GPE port	5	PCI
GPF port	16	CAI
GPG port	7	SDI
GPH port	10	SDI
GPI port	16	LCI
GPJ port	12	LCI
GPK port	16	Hos
GPL port	15	Hos
GPM port	6	Hos
GPN port	16	EIN
GPO port	16	Mei
GPP port	15	Mei
GPQ port	9	Mei

Fig 5.



a Port (GPP) 0

b Theoretically, pins can be as many as 32.

Fig 6.

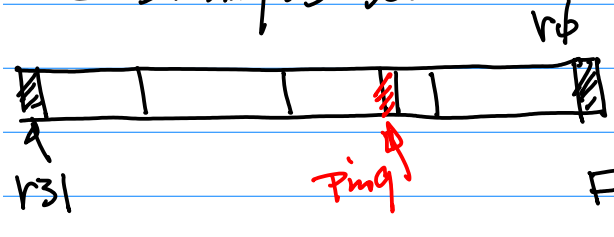


Fig 7

c. GPx DAT ping:

Root

GPxDAT[9]

d. Physical connector pin to GPxDAT[9] is given from SCH Design. Example, LPC1769

P0.9 → J2-5 (Connector J2, Pin5)

e. GPxPUD (Pull-up/Down) SPR.

f. Question: How many Control functions for a Single Control Register? Can we have

Homework: Down Load Image And Bring up your Target.

Oct. 13 (Wed)

Ref: 1° 202F-112 - Homework (Need Submission to CANVAS)

2° 2021F-113 - LPC17xx.ppt (For NXP LPC17xx platform)

Homework, Note GPIO pins selection Select One for Output, One for the Input, for "Hello, the world" program.

Turn on/off LED via (Input) GPIO pin

Read from GPIO pin for "1" or "0" when switch is toggled:

Note: for MCU Xpresso IDE down Load, you would have to Become a developer at NXP website, www.nxp.com. Then, Down Load MCU Xpresso, and finish the installation. Once installed, down Load LPC1769 pitch from Class github, import it into your IDE.

Example: Continued from GPxCON Number of Possible control functions?

From CPU Datasheet (ARM11), SPRs

Register	Address	RM
GPBCON	0x7F008020	RM
GPBDAT	0x7F008024	RM
GPBPUD	0x7F008028	RM
GPBCONSLP	0x7F00802C	RM
GPBPUDSLP	0x7F008030	RM

(1)

(2) memory (Address)

Define GPB0 as a input pin

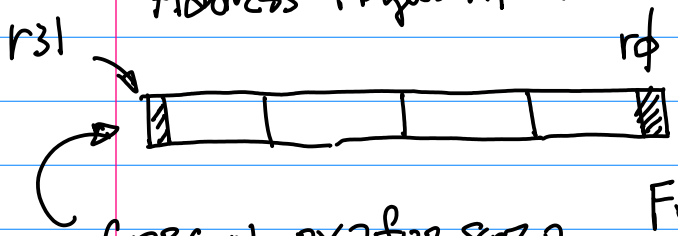
GPBCON	Bit	Description	
GPB0	[3:0]	0000 = Input	0001 = Output
		0010 = UART RXD[2]	0011 = Ext. DMA Request
		0100 = IrDA RXD	0101 = ADDR_CF[0]
		0110 = Reserved	0111 = External Interrupt Group 1[8]
GPB1	[7:4]	0000 = Input	0001 = Output
		0010 = UART TXD[2]	0011 = Ext. DMA Ack
		0100 = IrDA TXD	0101 = ADDR_CF[1]
		0110 = Reserved	0111 = External Interrupt Group 1[9]
GPB2	[11:8]	0000 = Input	0001 = Output

Table1

Note 1' 32 Bit Architecture

32 Bit SPRs (Special Purpose Register)

Address 4 Bytes Apart.



GPBCON 0x7f00_8020

Fig1.

GPB0 → pin0 → CPU, GPBDAT[0]

GPBCON[3:0] = 0000 // Define GPB0 input

GPBCON[3:0] = 0001 // Define GPB0 output

GPBCON[3:0] = 0010, UART Rx (Receiving)

UART Serial Communication

"2+1" pins minimum Requirements to establish UART Communication

- Rx (Receiving)
- Tx (Transmitting)
- GND

Connector Type, DB-9

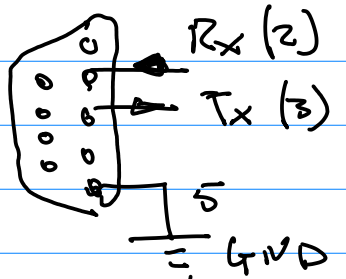


Fig.2

No. of possible functions for Control register:

$2^{32} = 4$ G.

Example: Implement/Design "Hello, the world" program.

Steps Involved for this is

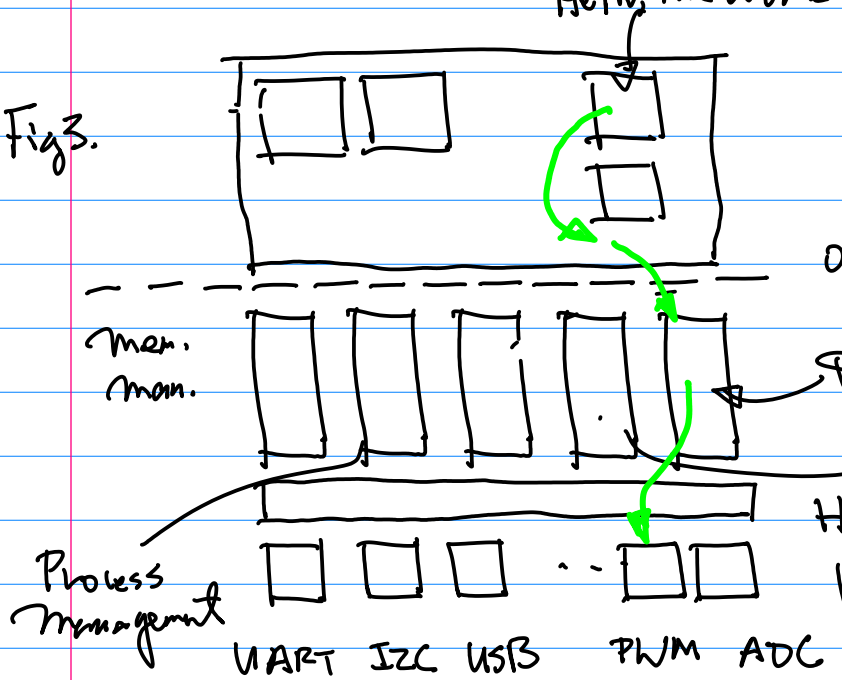
1°. Identify the pin(s) from a connector of a target Board;

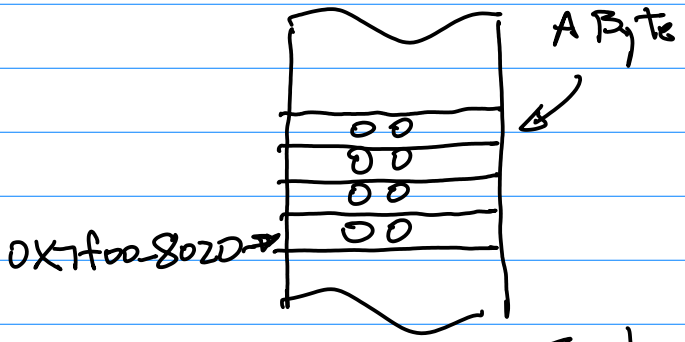
2° Find Driver Program to allow us to config GPP for Input/Output function.

Note Driver Program in the environment where O.S. is installed, can be accessed in a user Space.

From software

Write 0x0000_0000 to define GPB0 as an input to the following memory location.





A program will access to GPIO Devices as if it access to a file

- Open the file
- read from the file
- Write to the file

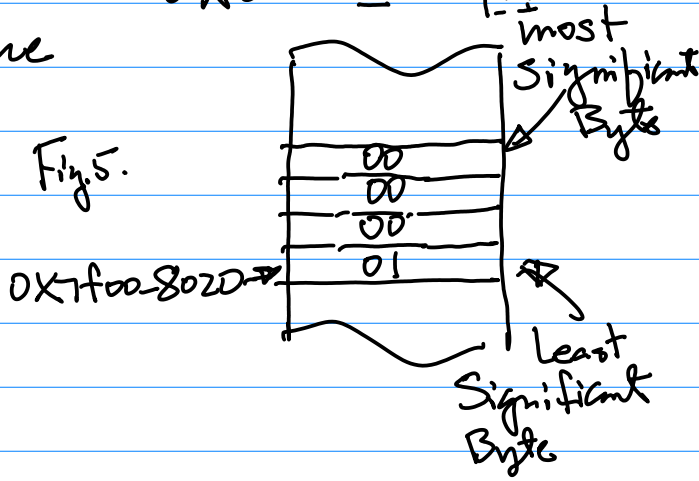
In user Space

In the kernel space, SFRs, like GPxCON, GPxDAT will fulfill the I/O I/F function.

Now, Define GPB0 as an Output, from CPU Datasheet

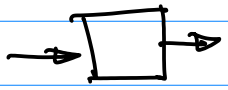
GPBCON[3:0] = 0001

Write 0x0000_0001 (Hex.)



Now, Discussion on IDE (MCU Xpresso)

Compiler



C/C++ High Level Language → Machine Code "0"s and "1"s

Cross-compiler

Compiler in a host machine which has different Architecture than the target platform.

CPU Architecture (Memory Mapping) → Cross Compiler

a. Machine code for Arm

b. fits to the target CPU

"Port"

Cross Compiler to a target CPU

Example: MCU Xpresso

GPIO program (Project) → firmware program w/o O.S.

Sample code for GPIO operation.

hello-world.c code

LPC_GPIO0 -> FIODIR

Target CPU | Peripheral Controller | Fast I/O | Direction

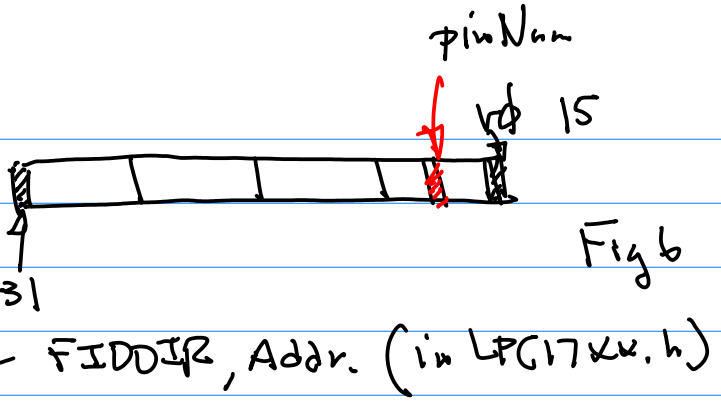


Fig 6

FIODIR, Addr. (in LPC17xx.h)

#define FIODIR 0xUUUU_UUUU

LPC_GPIO0 -> FIODIR

|= (1 << pinNum); ... (1)

for Example make P0.3 as output pin. From SCH.

Physical pin

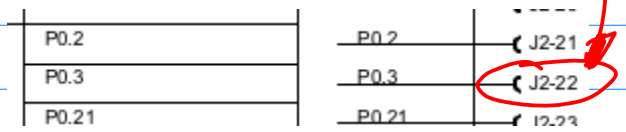
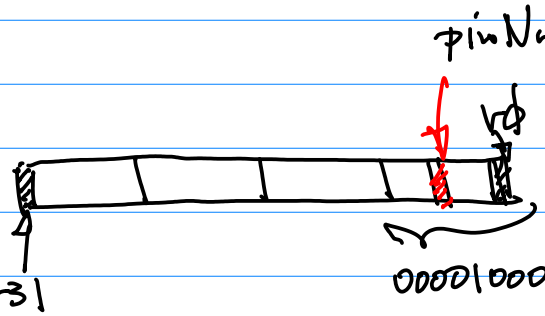


Fig 7.

"|=" bitwise OR

Write Binary Pattern in Fig 6. to set P0.3 as an output.

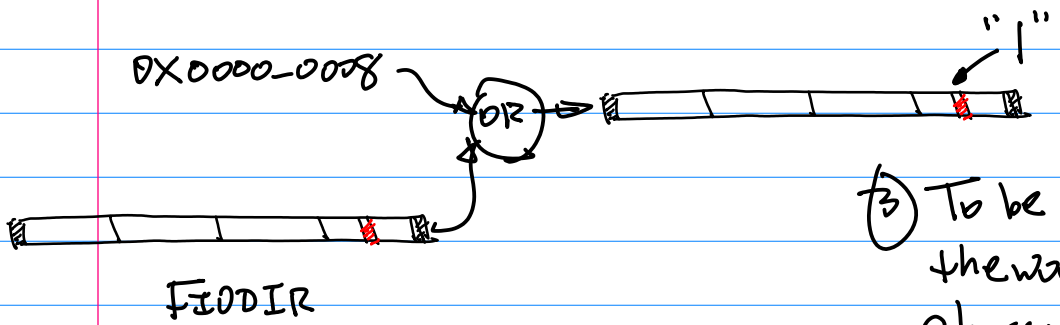


00 00 00 08

0X0000_0008

0X8

Fig 8



③ To be able implement "Hello, the world" program on your Chosen platform.

LPC_GPIO0->FIOSET

Special purpose Register: "DAT" register

Note: ≡ CPU Datasheet { ARM11 GPP, LPC GPP

≡ SPRs { ARM11: GPxCON, GPxDAT, GPxPUD; LPC: FIODIR, FIRSET

LPC_GPIO0->FIOSET = (1 << PinNum);

≡ Sample code LPC17xx.h

{ Peripheral Control mem. mapping

GPIO (Last 2~3 pages)

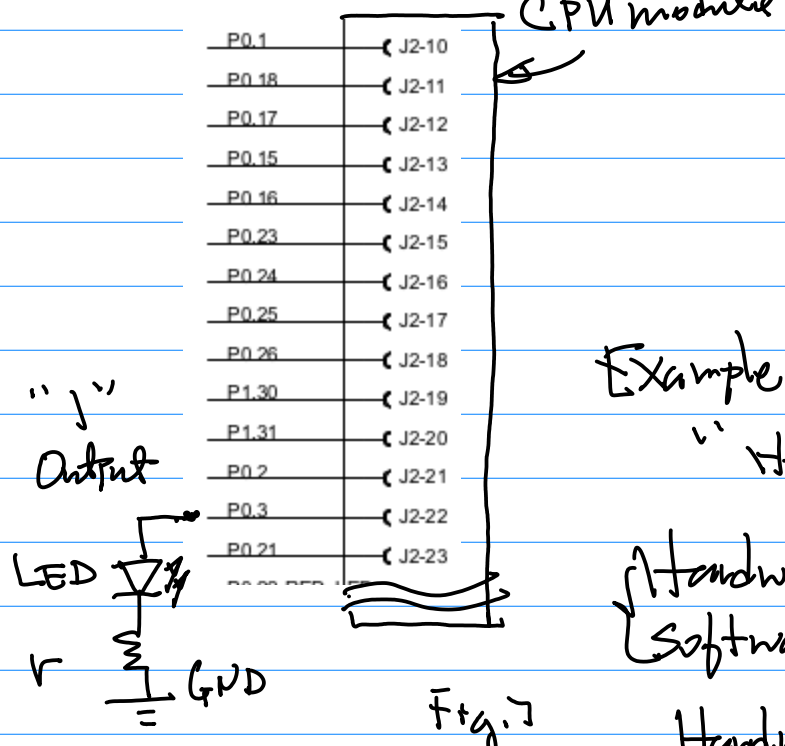


Fig.1

Example: Design/Implement "Hello, the world" program.

{ Hardware Design, Software Design.

Hardware Design.

Step1. Identify Hardware platform, Identify GPIO pins.

Requirements: Define Init & Config

① Pattern Based on CPU Datasheet

② Analyze SPRs responsible for GPIO operations.

To start the Design with the Sch of a chosen target platform.

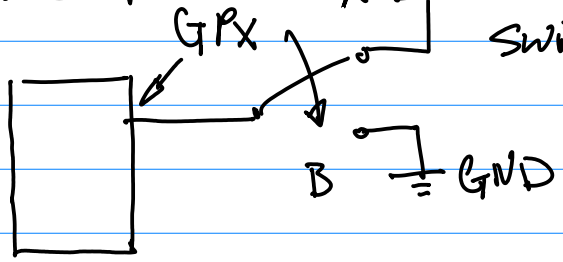
Find Connector(s) information

Identify GPIO pins

Step 2.

GPIO { Input { "1" / "0"
Output { "1" To turn on LED
"0" To turn off LED

Input CKT:



CPU (Connector) Fig. 8a

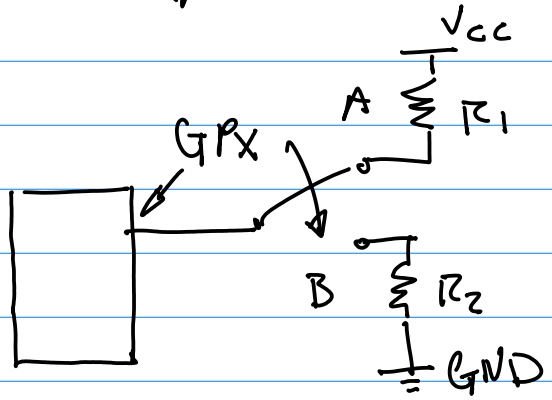


Fig. 8b $R_1 = R_2 \simeq 1K\Omega$

Note: 1° CMOS [0, 3.3V]

Logic "0" Logic "1"

$I_{CPU} \simeq 10mA$

For S/W connected to "A"

$$\frac{V_{CC}}{R_1} = I_{CPU} \quad \ldots (2)$$

$V_{CC} = 3.3V$, $I_{CPU} = 10mA$

$$\therefore R_1 = V_{CC}/I_{CPU} = \frac{3.3}{10\times10^{-3}}$$

$$= 3.3\times10^{2} = 330\Omega$$

For S/W @ B

Assum GPx is output high

$V_{GPx} = 3.3V$

$$I_{CPU} = \frac{V_{GPx}}{R_2} \quad \ldots (3)$$

$$R_2 = V_{GPx}/I_{CPU}$$

$$= 3.3/10\times10^{-3} = 330\Omega$$

Output Test.

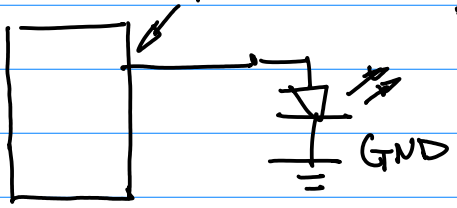


Fig. 9a

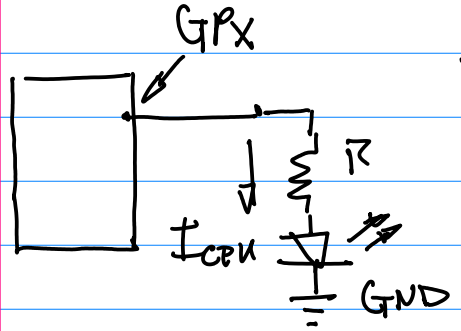
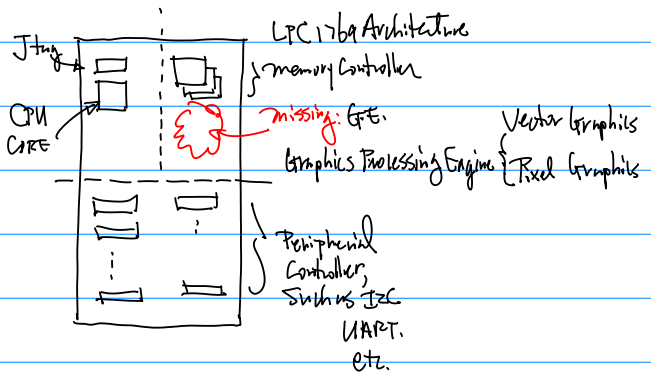


Fig 9b.



Let $I_{CPU} = 10mA$;

$V_{LED} = 1.2VDC$

$I_{CPU} \cdot R = V_{GPX} - V_{LED}$ ... (4)

$$R = \frac{V_{GPX} - V_{LED}}{I_{CPU}}$$

$$= \frac{3.3 - 1.2}{10 \times 10^{-3}} = 2.1 \times 10^{2}$$

$= 210\ \Omega$

Bring your Prototype Board for quick Demo.
(Target Board + Carrier Board
with I/O Testing Ckt)

Oct. 24th (Wed)

Today's Topics: 1° GPIO Interface Design On LPC/ARM11. 2° GPIO Implementation On target platforms (Jetson NANO, Pie)

Note: Homework Discussion

CMPE244

19

CMPE 244
Homework CPU Architecture and Sample IDE

1. Use LPC1769 CPU datasheet as an example to create a CPU block diagram based on the discussion in the class, e.g.,
1.1. Find CPU or SOC (System on Chip) block diagram of you chosen platform (Jetson NANO, Nvidia Jetson Tx2, or Pie) and copy it into word document.
1.2. Identify the GPP port controller (general purpose port block, e.g., GPIO block) in the block diagram;
1.3. Find the target board connector diagram and place this diagram in your word document. Then choose 2 GPIO pins, one for output and one for input, for the coming GPIO hello-the-world assignment.

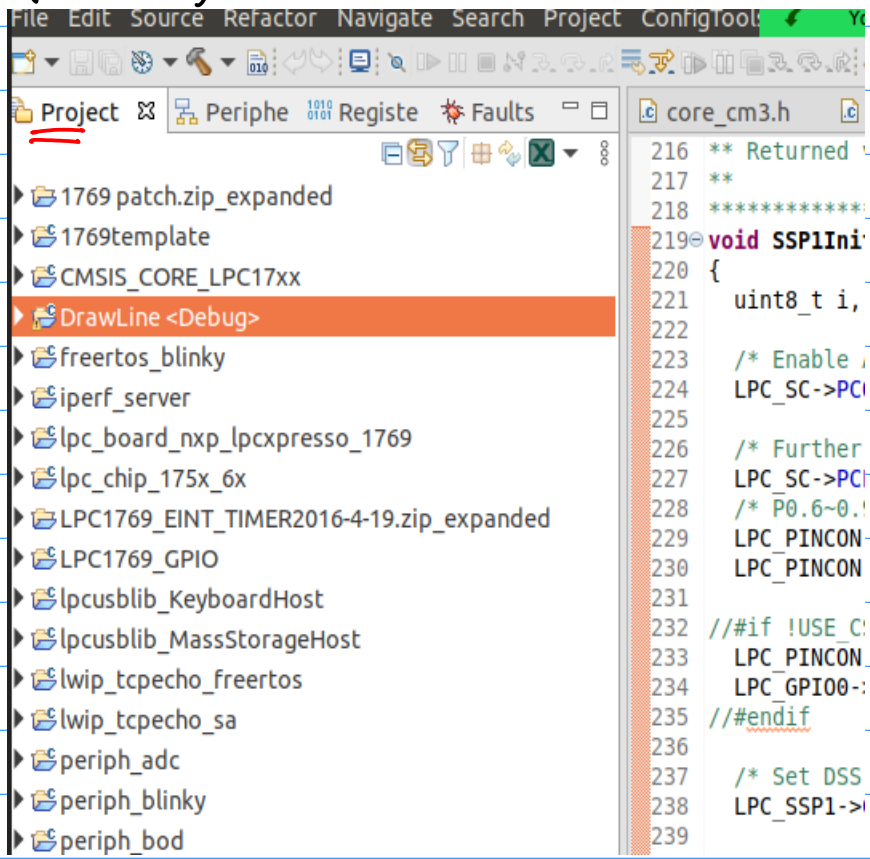
① finAssignment ② Controller from Architecture

2. Install NXP MCU Expresso and import LPC1769 patch. You can find the LPC1769 patch download from here
https://github.com/hualili/CMPE240-Adv-Microprocessors/blob/master/1769%20patch.zip

Zip to be imported to the IDE.

Project Panel, Import Project into your IDE

(1)



Import Sample project (.zip) into the IDE.

https://github.com/hualili/CMPE240-Adv-Microprocessors

Configuration of IDE to work with Class example on GPIO Interface.

Note: this IDE will allow us to analyze special purpose registers and their init & config, as well as mapping CPU architecture to memory map, and then making the connection to IDE cross compiler.

Once IDE + LFC17b9 patch Import is accomplished, then a+b will be posted on class github together with New Homework Assignment.

a. Import GPIO Simple project
b. Analyze + Read (Code Walk Through the GPIO Code Handouts)

Note: Please Purchase Wires, LEDs (2~5PCS), Resistors (10Ω~100KΩ). Need them for Next week Homework ("Hello, the world" on your target platform).

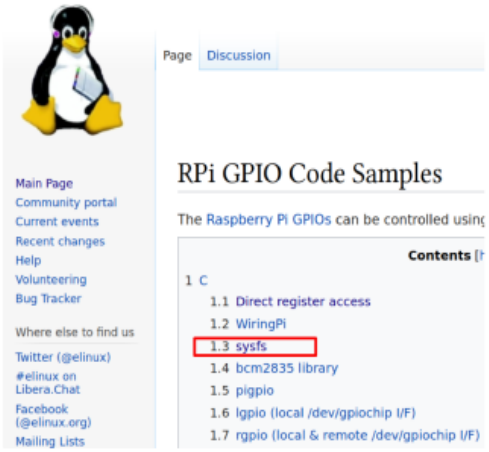
Jumper

Example: GPIO Implementation on NANO.

Ref: Ref1. Linux { NANO, Pie

Homework Due Oct 27th, Submission to CANVAS.

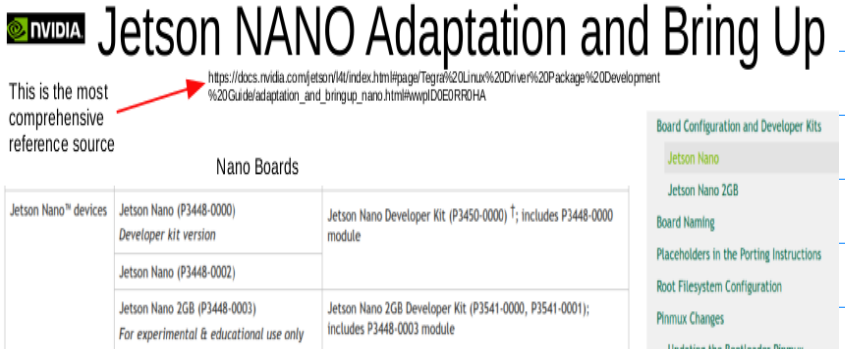
1. Bring up the target platform. e.g. Jetson NANO or Pie. (Submission of a photo. Showing System Setup and Screen of the target platform;



Ref2. from Nvidia Developer forum.

2. photo of your GPIO Circuit
   a. Input CKT: (S/W or Jumper Wire to Allow input "0" or "1")
   b. Output CKT: with LED on.
3. Submission
   3.1. photos in 2.
   3.2. Source code + Binary (Executable)
   3.3 Readme file.

please zip them into One file, Submit to SJSU CANVAS.



Step1. Down + Install Kernel Image (O.S.) Copy Kernel image to SD Card, then Bring

Step2. Identify connector(s) which provide

GPIO pins to Build Interface.

However GPIO Mapping is different

(1) 40 pin Connector is the same as the one for Pie

NVIDIA Jetson Nano J41 Header Pinout

https://www.jetsonhacks.com/nvidia-jetson-nano-j41-header-pinout/

(2) General Guideline

Note: I2C and UART pins are connected to hardware and should not be reassigned. By default, all other pins (except power) are assigned as GPIO. Pins labeled with other functions are recommended functions if using a different device tree.

1. take Pin 1 Vcc (3.3V) and Pin 39 GND to test out LED, make sure you can light up a LED with 220 Ohm resistor in series.

	GND	25	26	SPI_1_CS1	gpio20
	I2C_1_SDA *I2C Bus 0*	27	28	I2C_1_SCL *I2C Bus 0*	
gpio149	CAM_AF_EN	29	30	GND	
gpio200	GPIO_PZ0	31	32	LCD_BL_PWM	gpio168
gpio38	GPIO_PE6	33	34	GND	
gpio76	I2S_4_LRCK	35	36	UART_2_CTS	gpio51
gpio12	SPI_2_MOSI	37	38	I2S_4_SDIN	gpio77
	GND	39	40	I2S_4_SDOUT	gpio78

Sysfs GPIO	Name	Pin	Pin	Name	Sysfs GPIO
	3.3 VDC *Power*	1	2	5.0 VDC *Power*	
	I2C_2_SDA *I2C Bus 1*	3	4	5.0 VDC *Power*	
	I2C_2_SCL *I2C Bus 1*	5	6	GND	
gpio216	AUDIO_MCLK	7	8	UART_2_TX */dev/ttyTHS1*	
	GND	9	10	UART_2_RX */dev/ttyTHS1*	
gpio50	UART_2_RTS	11	12	I2S_4_SCLK	gpio79
gpio14	SPI_2_SCK	13	14	GND	
gpio194	LCD_TE	15	16	SPI_2_CS1	gpio232
	3.3 VDC *Power*	17	18	SPI_2_CS0	gpio15
gpio16	SPI_1_MOSI	19	20	GND	
gpio17	SPI_1_MISO	21	22	SPI_2_MISO	gpio13
gpio18	SPI_1_SCK	23	24	SPI_1_CS0	gpio19
	GND	25	26	SPI_1_CS1	gpio20

Harry Li, Ph.D

(3) physical pin Number from your Target Board, "even" Number on One Row, "odd" Number on the other. Enumeration of the pin Numbers starts from 1

Ground

	GND	25	26	SPI_1_CS1	gpio20
	I2C_1_SDA *I2C Bus 0*	27	28	I2C_1_SCL *I2C Bus 0*	
gpio149	CAM_AF_EN	29	30	GND	
gpio200	GPIO_PZ0	31	32	LCD_BL_PWM	gpio168
gpio38	GPIO_PE6	33	34	GND	
gpio76	I2S_4_LRCK	35	36	UART_2_CTS	gpio51

(4) Left part (col) GPIO mapping. e.g. gpio149 → Physical pin 29

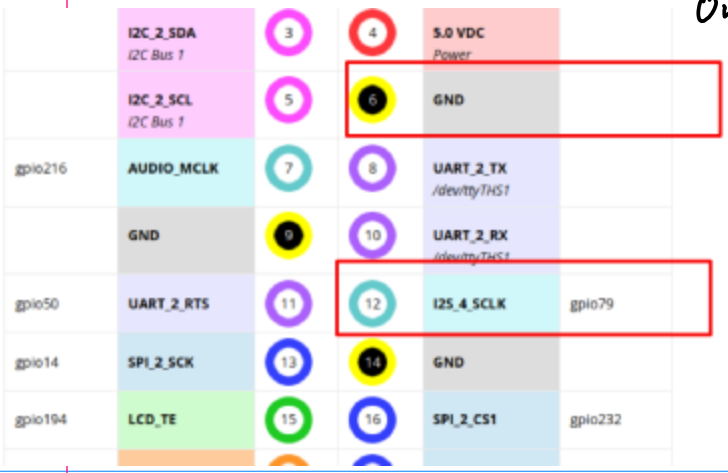
$echo 79 > /sys/class/gpio/export Which pin on my Connector corresponds to gpio79? → from the Pin-Assignment Diagram Pin12 is mapped to gpio79.

Step 3 Build the Hardware Prototype.

Circuit. Make Sure a 2 GPIO pins selected, one for Input, one for output; b Select the pins with Clear mapping information, e.g. CPU gpio v.s. physical pin connector.

Example: See Sample code below:

Build Connectivity Table.



CPU	J41	Note
gpio78	J41-40	line30/Input, Conn. To pin1 :3P3V
gpio79	J41-12	line32/Output Conn. LED+220Ω Resistor

Sample Code Part 2

```
28
29 //#define PIN  24 /* P1-18 */
30 #define PIN  78 /* P1-40 */
31 //HL: 2021-10-4 commented out #define POUT 4  /* P1-07 */
32 #define POUT 79  /*HL 2021-10-4 for 40 pin connector pin-12 */
```

① GPIO 78, physical pin 40 INPUT

Sample code Part 1. (Header) I took pie code for NAND

harry-nano@harry-desktop-nano: ~/harry/cmpe244/gpio-source

```
1  /*****************************************************************
2   * blink.c
3   *
4   * Raspberry Pi GPIO example using sysfs interface.
5   * Guillermo A. Amaral B. <g@maral.me>
6   *
7   * This file blinks GPIO 4 (P1-07) while reading GPIO 24 (P1_18).
8   */
9  //https://elinux.org/RPi_GPIO_Code_Samples#sysfs
10 // HL: 2021-10-8
11 // build with:
12 //  g++ -O1 -g -o mem gpiomem.cpp -Wall -std=gnu++17
13 // run with:
14 //  sudo ./mem
15
16 #include <sys/stat.h>
```

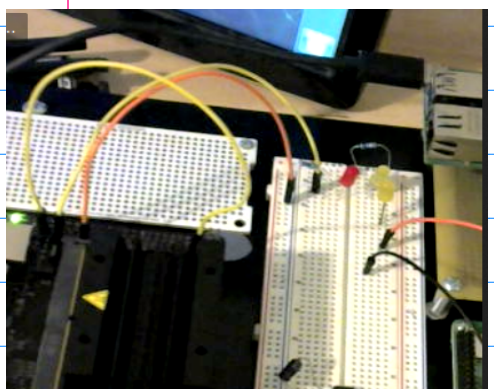
① ② ③ ④ ⑤

② GPIO 4 for Pie (Not Verified) P1-07 physical pin

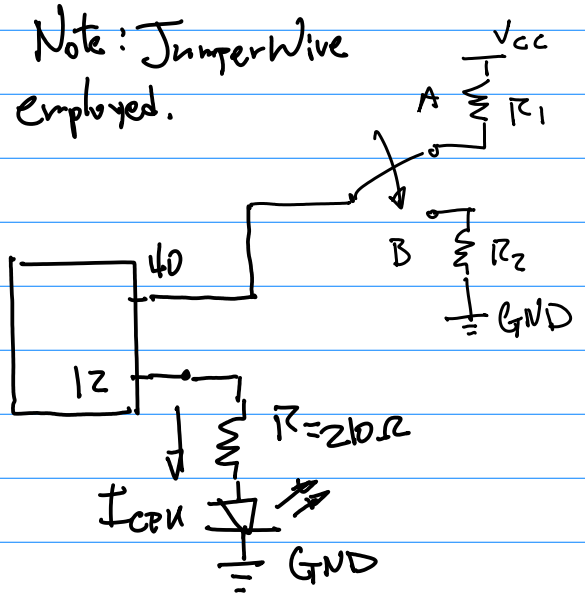
③ Ref.

④ From HL.

⑤ No Need.



Note: Jumper Wire employed.



Step 4. Software Implementation Preliminary Testing. CLI (Command Line Instructions)

CMPE244

```
$echo 79 > /sys/class/gpio/export
$ echo out > /sys/class/gpio/gpio79/direction
$echo 1 > /sys/class/gpio/gpio79/value
$echo 0 > /sys/class/gpio/gpio79/value
$echo 79 /sys/class/gpio/unexport
$cat /sys /kernel /debug/gpio
```

① Allocate gpio 79, used as a pair (export, unexport)

② Define it as output, direction

Note: please memorize these CLIs for the future use.

Once quick CLI Testing Done, go to C/C++ Implementation.

Example: Code Walk-Through

Name of the program gpio-pin.C to Be Changed.

1° line 1 ~ 14. Program Header (make a template)

Always provide program Header (For the Homework as well as)

Program Name:
Coded by:
Date & Version:
Status: (Debug, Release)
Copy Right:

Note: a Compilation & Build instruction
b Reference Source, such as github, URL etc.

2° line 29-32 info: gpio mapping pins

Need Pin Assignment Diagram.
See PPT. (github | ... | ... 114 ...) ppt.

① Static declaration of a function module

```
34 static int
35 GPIOExport(int pin)
36 {
37 #define BUFFER_MAX 3
38         char buffer[BUFFER_MAX];
39         ssize_t bytes_written;
40         int fd;
41
42         fd = open("/sys/class/gpio/export", O_WRONLY);
43         if (-1 == fd) {
44                 fprintf(stderr, "Failed to open export for writing!\n");
45                 return(-1);
46         }
47
48         bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
49         write(fd, buffer, bytes_written);
50         close(fd);
51         return (0);
52 }
```

② a. file system path for gpio device
b. open(...)

③

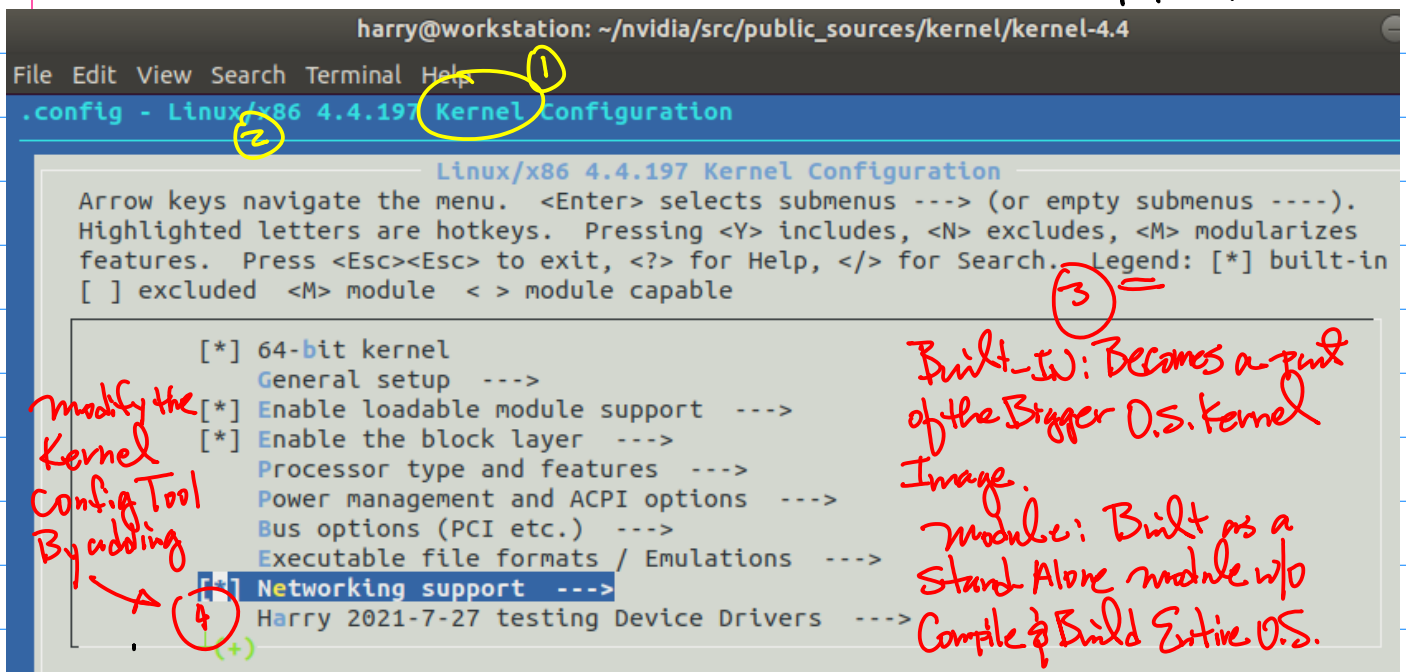
Consider Kernel Space Programming.
Device Driver Development, Debugging,
Improvement.

1. O.S. Source Distribution
   Target platform dependent distribution.
   Jetson NANO --- Nvidia Developer Forum
   Pie ... Manufacturer Related Site to Down Load Kernel Source Distribution.

2. Toolchain
   Cross-Compiler plus other Packages Provided by the Vendor for other functions, such as Computer Vision, GPU graphics, ML, AI, etc.

Arm11 as a base Line reference for Device Driver Development.
Tools for Device Driver Development IDE + CPU Architecture + Special Purpose Registers + O.S. modules as Device Driver.
Example: Introduction to Kernel Space programming.
In Particular, Device Drivers.

Example: ① make menuconfig, Kernel Configuration ② An x86 Host, for Linux kernel 4.4.197



CmpE 244 — Linux Device Driver — 25

Discussion on Sensor Interface and P.I.D. Controller Design.

Target Platform — LSM303 — stepper motor

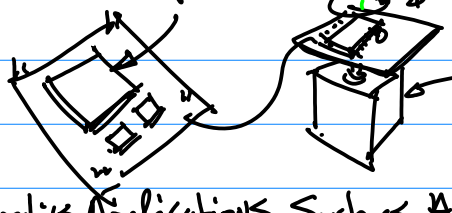


Fig. 2

Sensor Interface:

- X-Y-Z Acceleration Sensor LSM303; for Robotics Applications, Such as AGV, Autonomous Robot.
- Heart Beat/Rate sensor,

↑ In class, for Semester Long project

LSM303DLHC

Ultra compact high performance e-compass
3D accelerometer and 3D magnetometer module

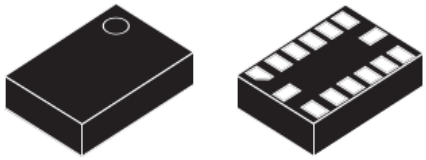
Preliminary data

X-Y-Z 3 Axis

Features

(1)
- 3 magnetic field channels and 3 acceleration channels
- From ±1.3 to ±8.1 gauss magnetic field full-scale
- ±2g/±4g/±8g/±16g selectable full-scale
- 16 bit data output

(2)
- I^2C serial interface



LGA-14 (3x5x1mm)

Fig. 2

Note: In Fig. 1. User Space program, P.I.D. Controller (Proportional, Integral, Derivative) → Device Driver (Kernel Space) → ⇐ Sensor: LSM303; ⇒ Actuate stepper motor.

Oct. 27 (Wed)

Today's Topics: 1° GPIO Homework Quick Review/Presentation of the homework. 2° O.S. Source Distribution & Toolchain, Example on Device Driver

Development. 3° Sensors Needed for Semester Long Project.

Note: Homework on LPC Reference Platform. IDE MCUXpresso, Requirements:

Requirements

1° Import LiCi76q patch → Set the Development Environment

2° Import GPIO project in Zip format, to exam/Investigate Special Purpose Registers, and their roles in Init & Config → Connection to Datasheet (800+ pages)

CPU Architecture pp.13?
Memory map
Special purpose Register

3° Firmware Concept.

Note: midterm schedule
Nov. 17 (Wed) 1 hr. Exam.
2~3 Question.

Example: Kernel Source Distribution & Toolchain Installation.

Objective: To be able to Build Kernel image, to write/Debug Device Drivers.

What Do we need:

1° O.S. Source Distribution from the Vendor of your target platform.

a. Base Line Reference: ARM11
b. NANO or c. Pie, Down Kernel Source Distribution from the Company.
(Jetson NANO)
please check the class github By the end of the Day today.
Pie.

2° Tool Chain, Very often is packaged together with the Kernel source distribution, for Example for Jetson NANO. (L4T) "Linux (O.S) for Tegra (GPU platform)

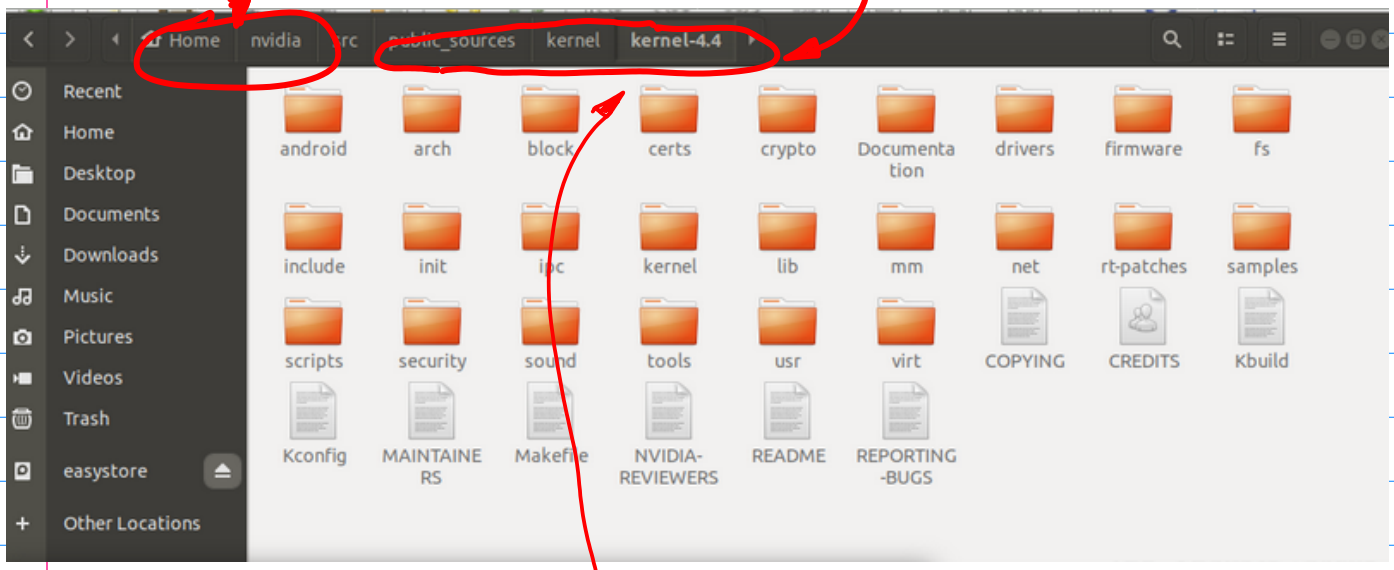
Step 1. Down Load Kernel Source And Tool Chain. And Install the Kernel Source & Tool Chain
(A Part of Homework Due Next Week Nov. 3rd)

Down Load Instruction & Installation Guide are given by the vendor.
for LINUX (Ubuntu 18.04) Host — laptop.

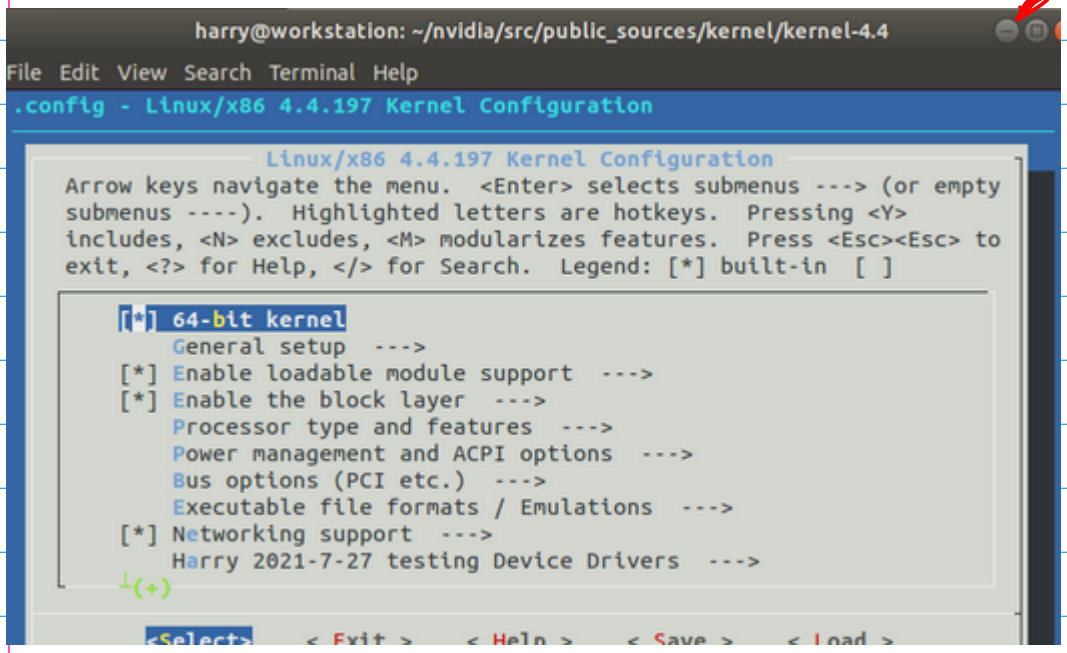
To unzip the Down Load file.
(un-tar) $tar -xzvf → O.S. Source

See Screen Capture Next Page.

① Installed O.S. Source + Toolchain ② Toolchain, make menuconfig.



Step 2. Execute $make menuconfig. Once the DownLoad is un-zipped, then go to the proper directory. for Example /kernel/kernel-4.4 (Depending on which version of the Distribution you have). Then to start $make menuconfig.



Homework: Due A week from today (Nov. 3rd) on CANVAS.

Submit: 1. Screen Capture of make menuconfig

2. Write Readme to Describe download, unzip, installation Process

3. Provide Version, URL and clear description of the O.S. Down Load.

Example: make menuConfig on ARM-11 platform.

Background: Programming in Kernel Space.

PART II of the Homework: Purchase a
1. Stepper motor Drive (PWM Device Driver — PID Controller User Program Implementation), And a 2. Stepper Motor, NEMA17.
(Have them Ready By Next week Nov. 3rd)

```
root@harry-laptop:/opt/FriendlyARM/mini6410/linux# ls
arm-qte-4.7.0           busybox-1.17.2  rootfs_qtopia_qt4    x86-qte-4.6.1
arm-qt-extended-4.4.3   examples        rootfs_qtopia_qt4-s  x86-qt-extended-4.4.3
arm-qtopia              linux-2.6.38    u-boot-mini6410      x86-qtopia
root@harry-laptop:/opt/FriendlyARM/mini6410/linux#
```

③ Program Examples in User Space to interface to Kernel Space Device Drivers

① Kernel Source

② u-boot for the Referene

Now, Consider User Space program(s) which requires the utilization of Device Driver in Kernel Space.

④ Camera Interface

① GPIO LEDs

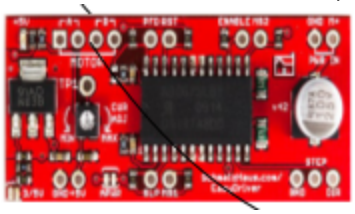
```
root@harry-laptop:/opt/FriendlyARM/mini6410/linux/examples# ls
adc-test  camtest  i2c       lecSSBike  leds        readme.txt     threadTest  vfp-test  www
buttons   fft      lecPID    led-player  pwm         robotControl   usbcam      Vision
```

③ User-Created Example

② PWM — pulse width modulation. to Drive Stepper Motor

⑤ i2c EEPROM (Electronic E. Progr. Read Only Memory)

~ Invoma



Stepper motor Drive ~$10

Other Stepper motor Drive can be used aswell.

User Space Program To Kernel Space (Device Driver), GPIO (Program: leds.c)

```
15          printf("hello\n"); //Feb. 16, 2015
16
17          if (argc != 3 || sscanf(argv[1], "%d",
18                          on < 0 || on > 1 || le
19                  fprintf(stderr, "Usage: leds l
20                  exit(1);
21          }
22
23          fd = open("/dev/leds0", 0);
24          if (fd < 0) {
25                  fd = open("/dev/leds", 0);
26          }
27          if (fd < 0) {
28                  perror("open device leds");
29                  exit(1);
30          }
31
32          ioctl(fd, on, led_no);
33          close(fd);
34
```

Ready Build Kernel Image, which has this Driver

① fd (file descriptor) is defined as a file with open(, , 0)

path (folder) and the device driver

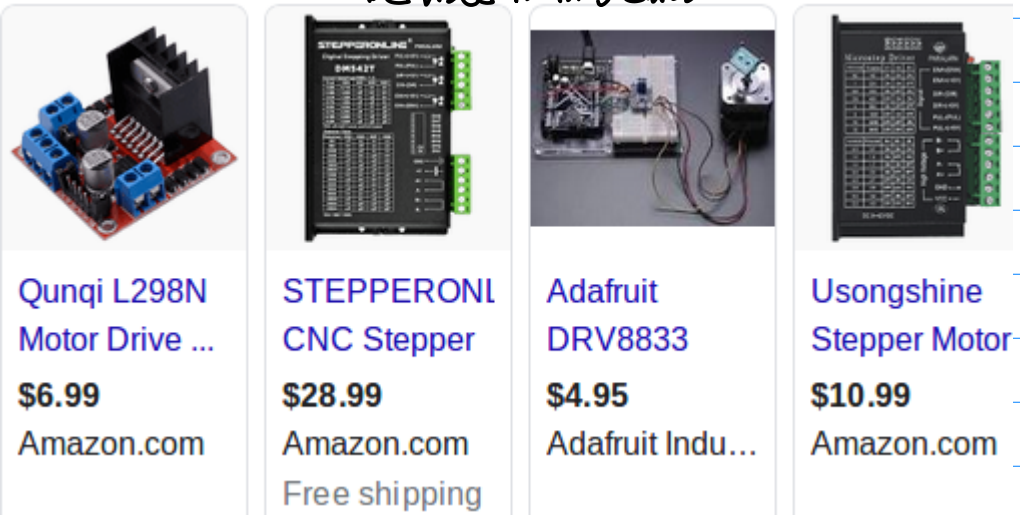
Required, please memorize!

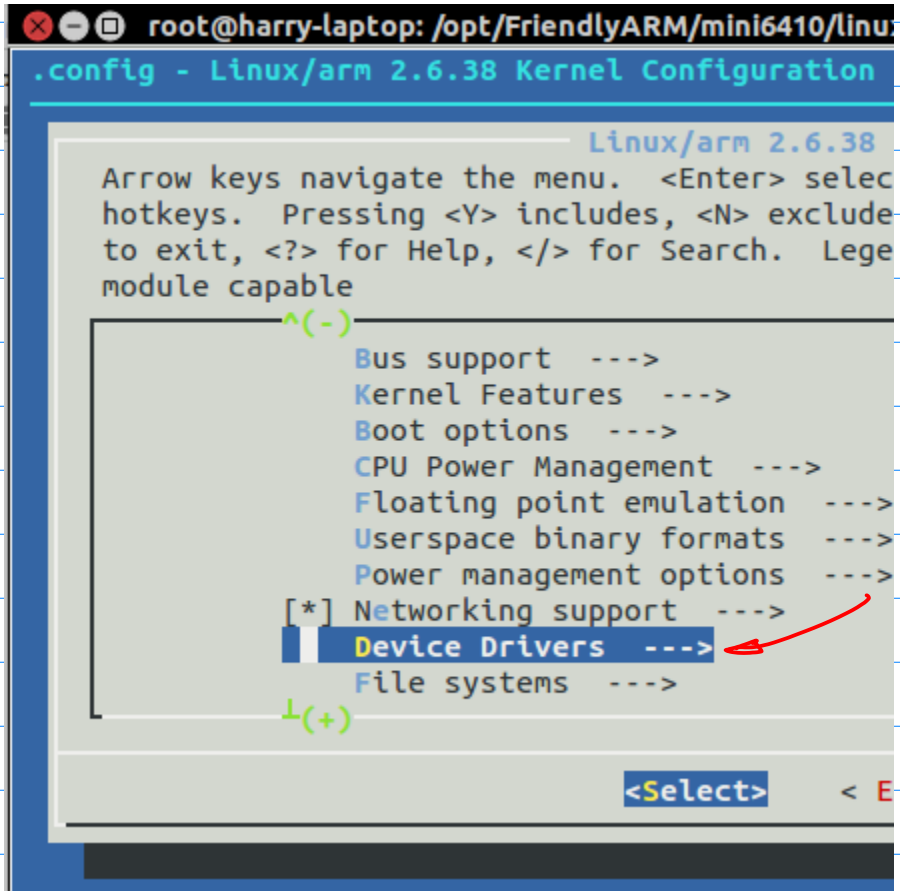
② ioctl(fd, P1, P2); function to communicate to device driver defined by fd. Required, Memorize!

③ Once done, close it. please! open–close

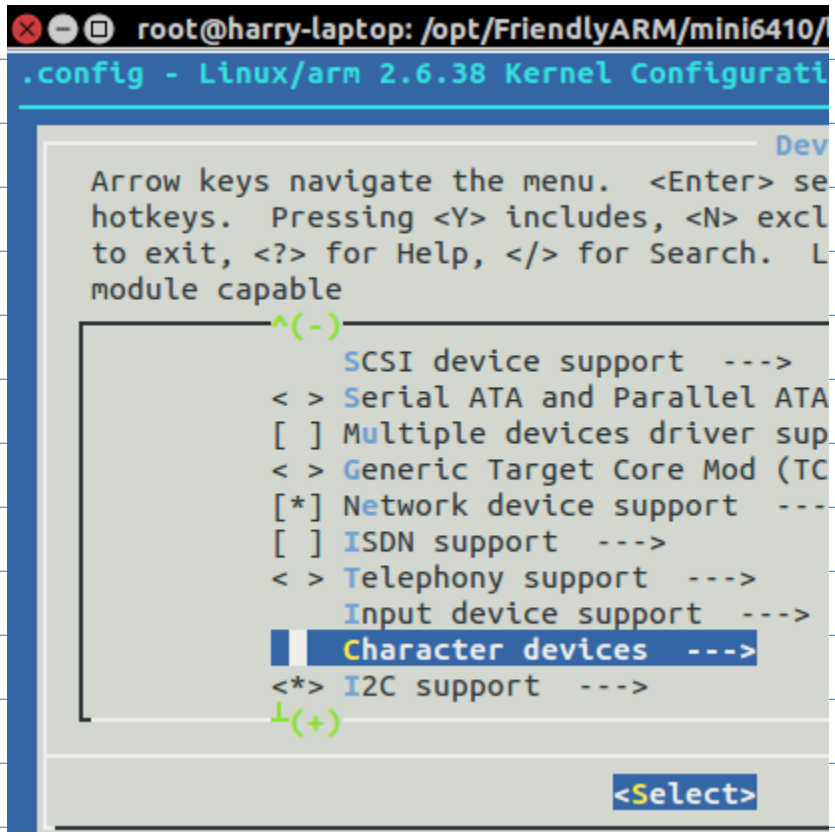
In Kernel Space.

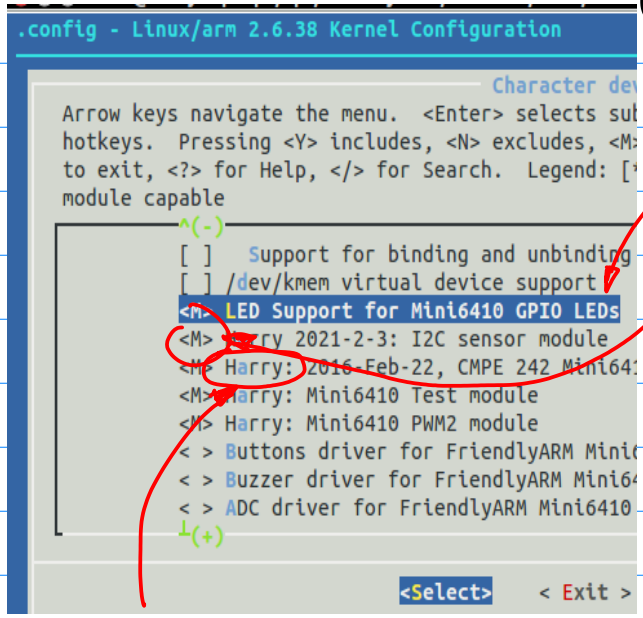
Note: The following Stepper motor Drive can be used in this class.





Select
(1) Go to Device Drivers
(2) Select "Char" Device
e.g. Character Device.





(3) GPIO Device Driver to turn on/off LED

Use KeyBoard "Space Bar" to select Build it into An Entire O.S. Kernel Image or Build stand-Alone module "M" without Building the entire Kernel O.S. And Later this module can be installed as a part of a kernel image.

or De-Select this module, e.g. Not Build

(4) User defined UI, By modifying the corresponding "Kconfig" file.

(Kernel)

Now, for Kconfig at each level, now the Kconfig@Char Device level

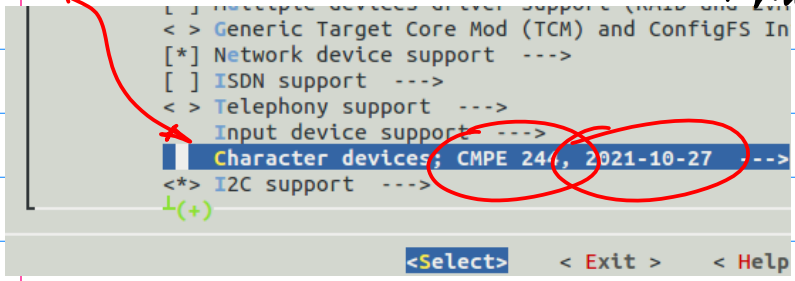
Now, Check "Kconfig"@Char told

(1)

(2)

UI will show this on the screen.

(3) User can make change of Kconfig, to Add/Remove/modify...



CMPE244

Defines select as a kernel, or select as "M" or De-select.

Keywords ① "config" ④ "tristate" ⑤ "depends on" Defines target 32 platform (with target CPU) ⑥ "default"

```
100 config MINI6410_LEDS
101         tristate "LED Support for Mini6410 GPIO LEDs"
102         depends on CPU_S3C6410
103         default y
104         help
105           This option enables support for LEDs connected to GPIO lines
106           on Mini6410 boards.
107
```

⑦ "help" verbage for help message.

② Connection from "Kconfig" file to Actual Device Driver (/char), note the folder (Path)

```
/opt/FriendlyARM/mini6410/linux/linux-2.6.38/drivers/char
mini6410_pwm_mod.c                    rocket_int.
```

~/drivers/char

③ Device Driver for "LED" in the folder /char.

```
-rw-r--r-- 1 root root 10696 Oct 27 15:48 mini6410_hello_mod
-rw-r--r-- 1 root root  1920 Feb 22  2016 mini6410_leds.c
-rw-r--r-- 1 root root 76847 Oct 27 15:48 mini6410_leds.ko
```

Source code

Compiled module, Ready to be used.

Copy this ".ko" file, for example, onto USB-stick, plug the USB into the target platform (e.g. ARM-11) then, Copy this file to your working directory, use

$ insmod ——— Name of the file

to make it a part of Kernel image ready to be used.

```
root@harry-laptop: /opt/FriendlyARM/mini6410/linux/linux-2.6.38
.config - Linux/arm 2.6.38 Kernel Configuration
                    LED Support for Mini6410 GPIO LE
CONFIG_MINI6410_LEDS:

This option enables support for LEDs connected to GPIO lines
on Mini6410 boards.

Symbol: MINI6410_LEDS [=m]
Type  : tristate
Prompt: LED Support for Mini6410 GPIO LEDs
  Defined at drivers/char/Kconfig:100
  Depends on: CPU_S3C6410 [=y]
  Location:
    -> Device Drivers
      -> Character devices
```

Requirements: ① To be able to understand UI and to locate Character Device Driver

② To be able to modify "Kconfig" file at \drivers\char

③ To be able use/choose keywords, 5 of them, listed Above. please memorize them!

"Kconfig" @ /drivers

root@harry-laptop: /opt/FriendlyARM/mini6410/linu

```
menu "Device Drivers"

source "drivers/base/Kconfig"

source "drivers/connector/Kconfig"

source "drivers/mtd/Kconfig"

source "drivers/of/Kconfig"

source "drivers/parport/Kconfig"

source "drivers/pnp/Kconfig"

source "drivers/block/Kconfig"
```

"Kconfig"

① connector, Reports Process event

root@harry-laptop: /opt/FriendlyARM/mini6410/linux/linux-2.6.38/drivers/connector

```
menuconfig CONNECTOR
        tristate "Connector - unified userspace <-> kernelspace linker"
        depends on NET
        ---help---
          This is unified userspace <-> kernelspace connector working on top
          of the netlink socket protocol.

          Connector support can also be built as a module.  If so, the module
          will be called cn.

if CONNECTOR

config PROC_EVENTS
        boolean "Report process events to userspace"
        depends on CONNECTOR=y
        default y
        ---help---
          Provide a connector that reports process events to userspace. Send
          events such as fork, exec, id change (uid, gid, suid, etc), and exit.

endif # CONNECTOR
~
```

Now, "Kconfig" at Kernel Level.

root@harry-laptop: /opt/FriendlyARM/mini6410/linux/linux-2.6.38

```
#
# For a description of the syntax of this configuration file,
# see Documentation/kbuild/kconfig-language.txt.
#
mainmenu "Linux/$ARCH $KERNELVERSION Kernel Configuration"

config SRCARCH
        string
        option env="SRCARCH"

source "arch/$SRCARCH/Kconfig"
```

① Kernel Build Docs

Example: A Simple Device Driver, Does nothing. But "print"

ADC: Analog to Digital Conversion ① "Hello" Driver GPIO Drivers PWM

```
root@harry-laptop:/opt/FriendlyARM/mini6410/linux/linux-2.6.38/drivers/char# ls mini6410_*
mini6410_adc.c       mini6410_hello_module.c      mini6410_leds.c      mini6410_pwm2.c
mini6410_adc.mod.c   mini6410_hello_module.ko     mini6410_leds.ko     mini6410_pwm2.mod.c
mini6410_adc.o       mini6410_hello_module.mod.c  mini6410_leds.mod.c  mini6410_pwm.c
mini6410_buttons.c   mini6410_hello_module.mod.o  mini6410_leds.mod.o  mini6410_pwmHarry.c
mini6410_buttons.o   mini6410_hello_module.o      mini6410_leds.o      mini6410_pwm.mod.c
root@harry-laptop:/opt/FriendlyARM/mini6410/linux/linux-2.6.38/drivers/char#
```

EXINT: External Interrupts Drivers

Note: "Hello" Driver. ① <linux/kernel.h> <linux/module.h>

② Syntax: static int __init target(void)

root@harry-laptop: /opt/FriendlyARM/mini6410/linux/linux-2.6.38/d

```
1  #include <linux/kernel.h>
2  #include <linux/module.h>
3
4
5  static int __init mini6410_hello_module_init(void)
6  {
7      printk("Hello, Mini6410 module is installed !\n");
8      return 0;
9  }
10
11 static void __exit mini6410_hello_module_cleanup(void)
12 {
13     printk("Good-bye, Mini6410 module was removed!\n");
14 }
15
16 module_init(mini6410_hello_module_init);
17 module_exit(mini6410_hello_module_cleanup);
18 MODULE_LICENSE("GPL");
```

① ② ③

③ static void __exit target Name

Steps to Test out this "Hello" module.

5-Step Process for Executing "Hello" module (In kernel Space).

Sensor Selection for Semester Long, Please discuss with your team member, And Selection By Next Wed

2021F-116b-heart-beat-...

2021F-117-#lec4-Stepp...

3D Accelerometer and 3D Magnetomete LMS303

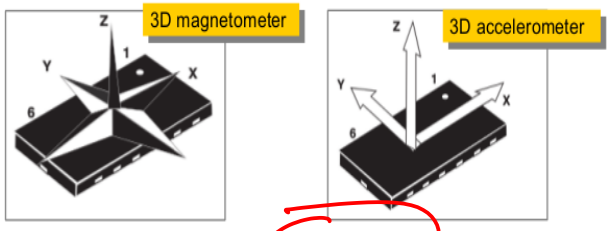


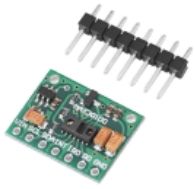
Table 9

I2C Interface

Max30100 Sensor

max30100 pulse oximetry and heart-rate monitor sensor

I2C-Compatible Interface



I2C I/F

https://www.amazon.com/MAX30102-Detection-Concentration-Compatible-Arduino/dp/B07ZQNC8XP/ref=asc_df_B07ZQNC8XP/?tag=hyprod-