Embedded Saftware
CMPE244

Sept. 29 (Wed) 4:00-8:00pm.
Zoom Link To Be Used for the Entire Semester

Harry Li, Office: Engr. Building Rm 269A
E-mail: hua.li@sjsu.edu.
Text messages (650)400-1116.

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 I.D.E (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

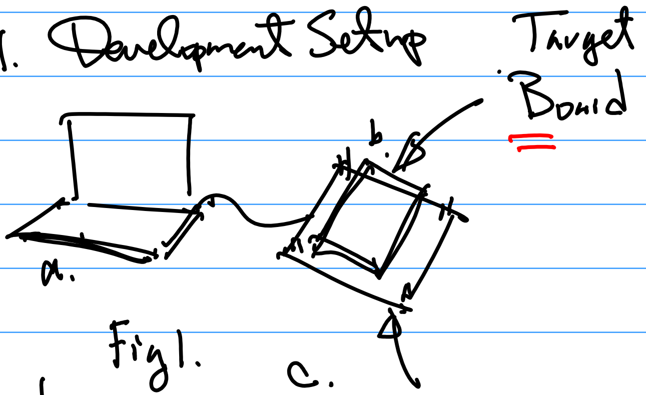


Fig 1.

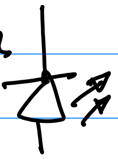
a. Host PC | Laptop, Linux Ubuntu 18.04
 Virtual Box Installed, then install Linux on top of it.

b. Target platform (To Be determined)

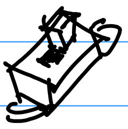
c. Wire Wrapping Board
 ① ~$3\frac{1}{2}$" × 4" physical Dimension
 Through Holes with metal Coating.

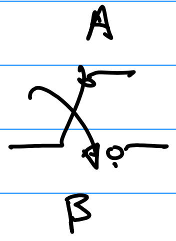
② 4 mounting Holes @ the corners, 4 stand-offs (Legs)

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



④ Toggle switch





2. OS. Architecture

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

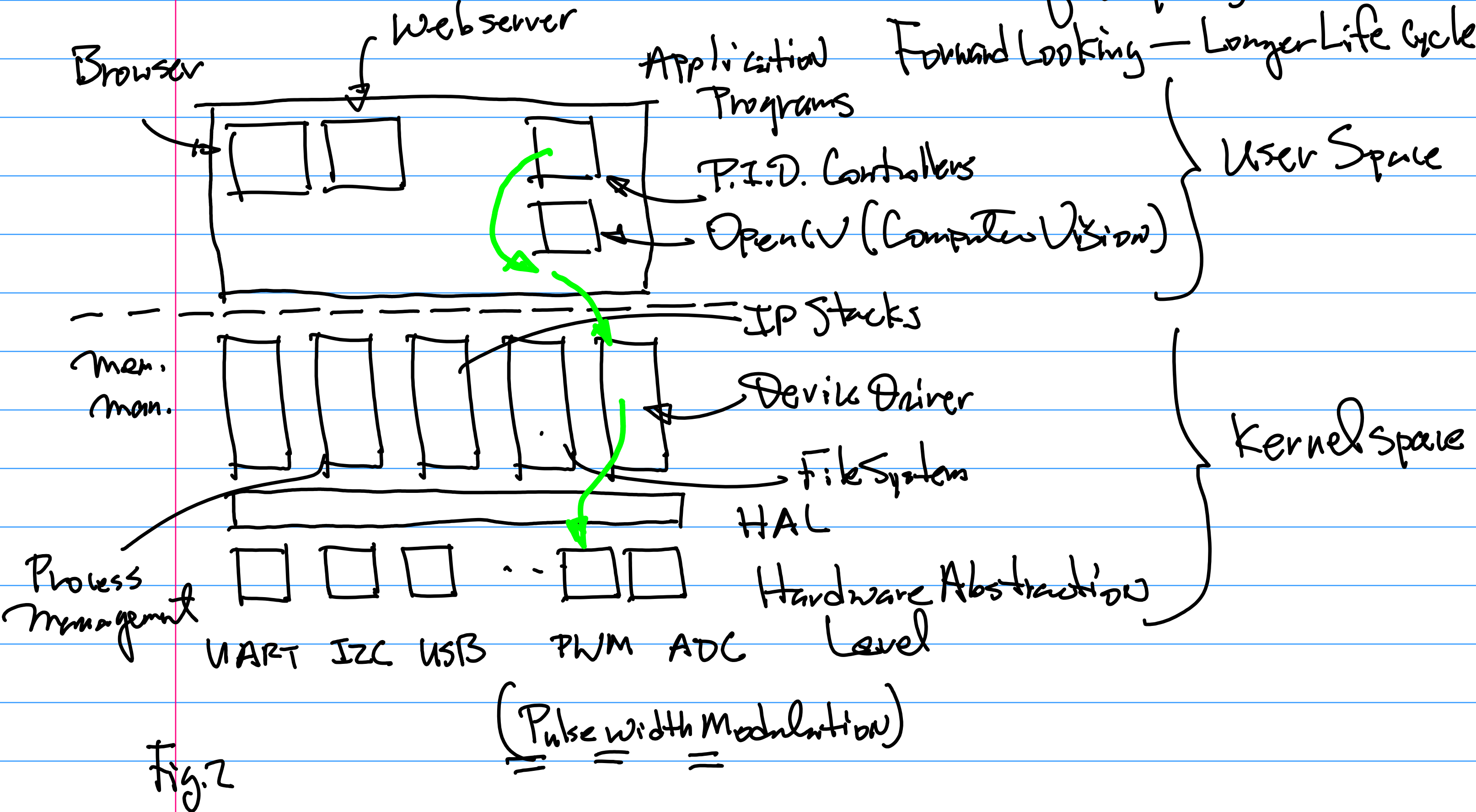


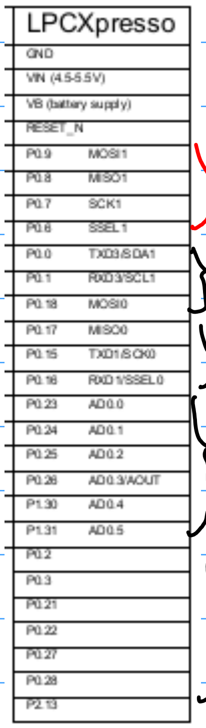
Fig. 2

Target Platforms To Consider

1. NXP LPC17xx, 1769
   Clock Rate: ~100MHz — 400MHz
   RTOS But Not Unix O.S.
   No Linux
   Rich I/O Interface
   Ref: git ~ 2021F-107b— . .

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

Example: SCH of LPC1769.



LPCXpresso	
GND	
VIN (4.5-5.5V)	
VB (battery supply)	
RESET_N	
P0.9	MOSI1
P0.8	MISO1
P0.7	SCK1
P0.6	SSEL1
P0.0	TXD3/SDA1
P0.1	RXD3/SCL1
P0.18	MOSI0
P0.17	MISO0
P0.15	TXD1/SCK0
P0.16	RXD1/SSEL0
P0.23	AD0.0
P0.24	AD0.1
P0.25	AD0.2
P0.26	AD0.3/AOUT
P1.30	AD0.4
P1.31	AD0.5
P0.2	
P0.3	
P0.21	
P0.22	
P0.27	
P0.28	
P2.13	

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

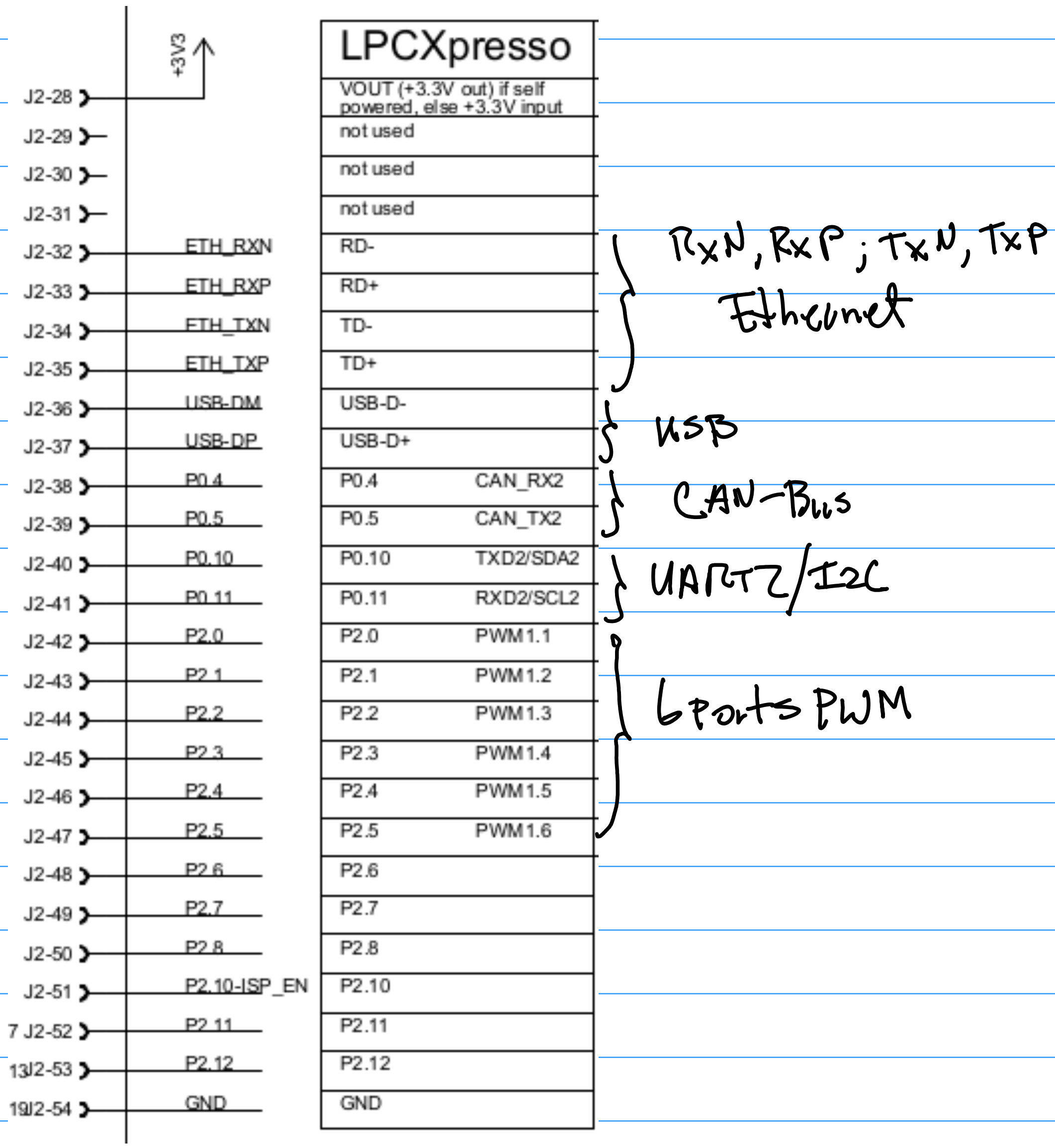
UART3 { Tx Transmission / Rx Receiving / multiplexed
SDA: Serial Data
SCL: Serial CLK
I2C

SPI0 / UART1

6 ADC (AD — Analog to Digital Conversion)

GPP (GPIO: General Purpose Port I/O)

CMPE244



Option Target platform: FPGA. Future Electronics.

FPGA Igloo2: RISC-V.

Superset of ARM Architecture

IP Core: Open Source. Supports RTOS

Limitations: No Unix/Linux O.S.

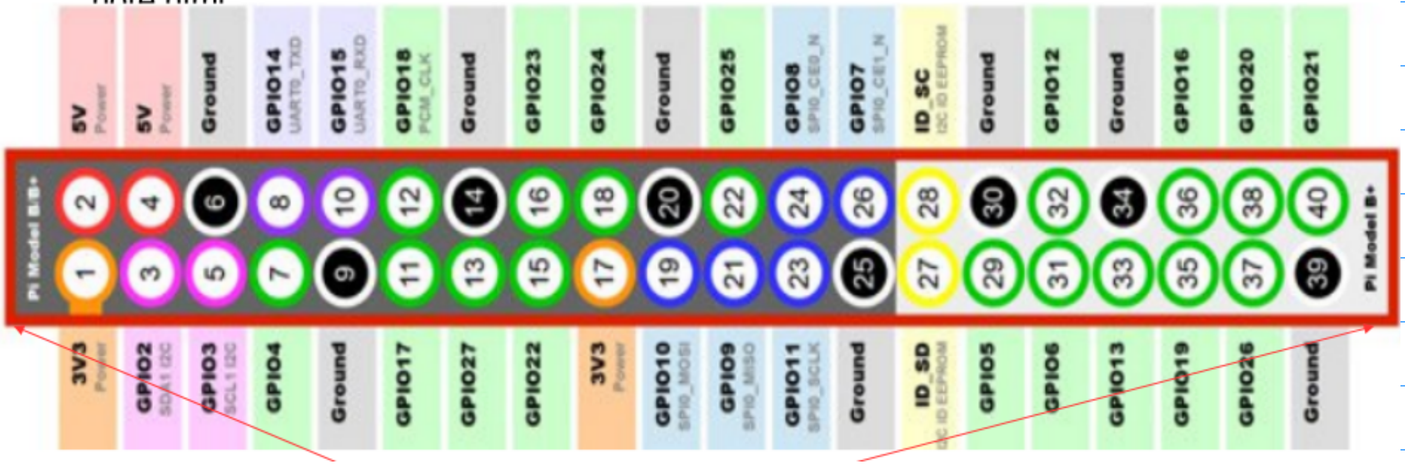
Smaller Gate Counts. → less Computational Capability.

Pie. BroadCom BCM

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



Pins: 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

CMPE244

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 Wire Wrapping 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

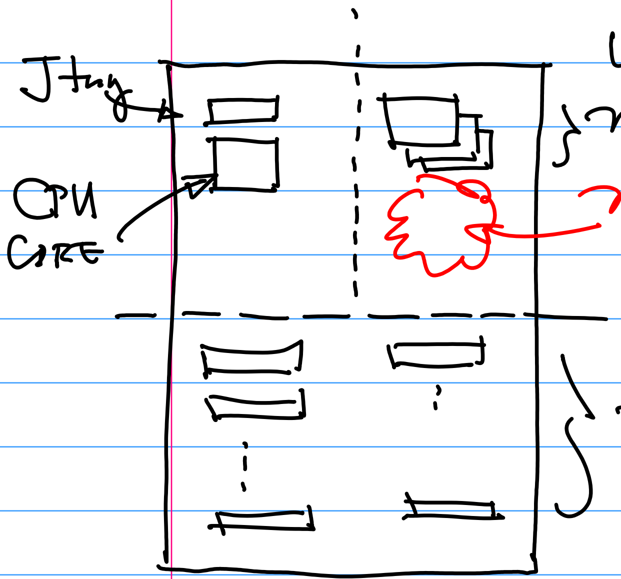
Sign up as a developer at nxp website
www.nxp.com
MCU Xpresso

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 Base Line
2) Samsung ARM11 Datasheet
CPU Block Diagram
memory map
Special Purpose Register
Tool Chain / Software Design, Implementation

CMPE244

LPC1769 Architecture



memory Controller

Graphics Processing Engine { Vector Graphics / Pixel Graphics

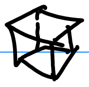
Peripherial Controller, Such as I2C UART. etc.

Fig1.

Fig.

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

Vector Graphics Engine { 2D Graphics { a. Display Device Driver / b. 2D primitive Graphics Processing / c. 2D Transformation { line / Circle
3D Graphics



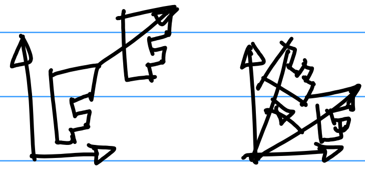
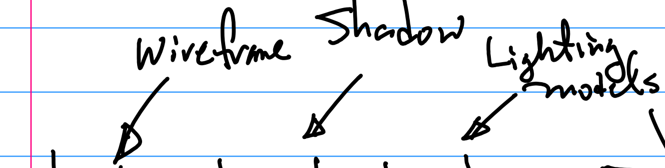


Fig.2b

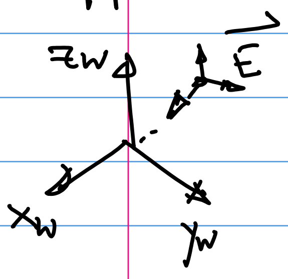


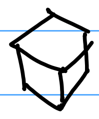
Ambient Light + Diffuse Reflection Specular Reflection

Transformation Pipeline

Texture mapping

Hidden Line / Surface Removal





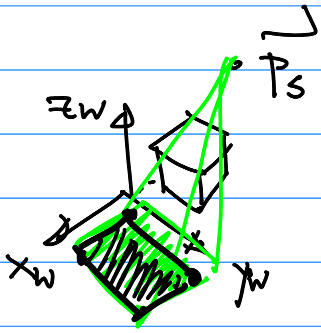


Fig.2

Memory map:

1. RISC: Reduced Instruction Set Architecture

1979 David Patterson

1982 John Hennessy

- Uniformity
- Regularity
- Orthogonality

2. 32 Bit RISC Architecture

Data Bus: 32bit, Bi-directional

$$D[m:n] \quad \text{Vector Notation}$$

Most Significant Bit — Least Significant Bit

$D[31:\phi]$

Endian "Little Endian"

Address Bus: 32bit, 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 fullfill 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?

Since 32 Bit Architecture → Addr. Bus 32bits

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

$2^{10} \ldots 1K$

$2^{20} \ldots 1K \times 1K = 1$ Meg.

$2^{30} \ldots 1M \times 1K = 1G$.

4 GByte (Byte Addressable)

CmpE244

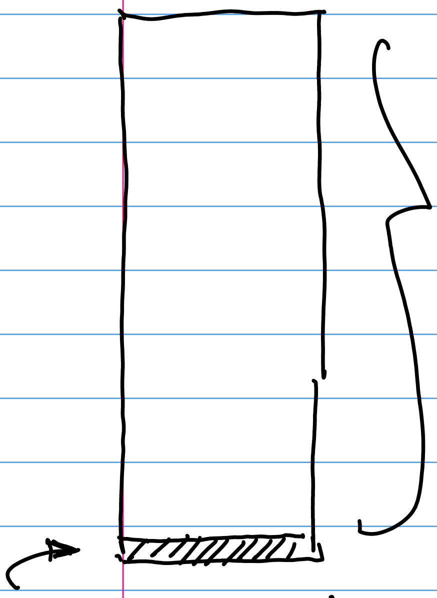


Fig. 3

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 ϕ
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 : ϕ]

$a_{31} a_{30} a_{29} \cdots a_2 a_1 a_0$

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	BANK ϕ
0 0 1 ⋮ 0	0X2000_0000	Bank 1
0 1 0 ⋮ 0	0X4000_0000	Bank 2
⋮		⋮
1 1 1		Bank 7

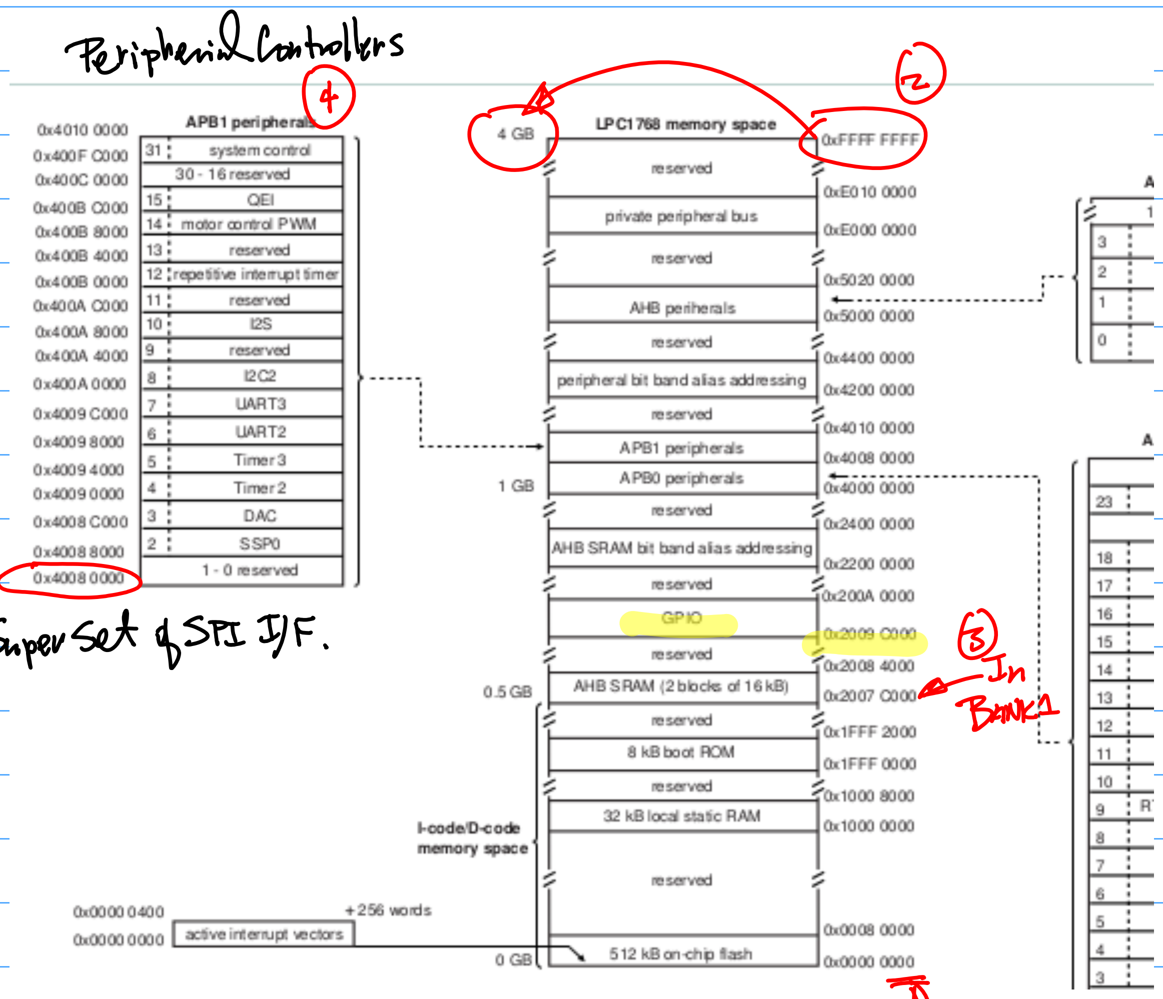
Datasheet, pp.14 from NXP LPC1769

memory map, SSPϕ as an example.
then, starting Addr. for memory Banks,

Starting Address SSPϕ:
0X4008_8000

Question: How much memory does SSPϕ need?

Peripheral Controllers



SSP0: Super Set of SPI I/F.

Fig 4

1 PWR-up Addr

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, e.g. GPIO)

3 Common Types of SPR

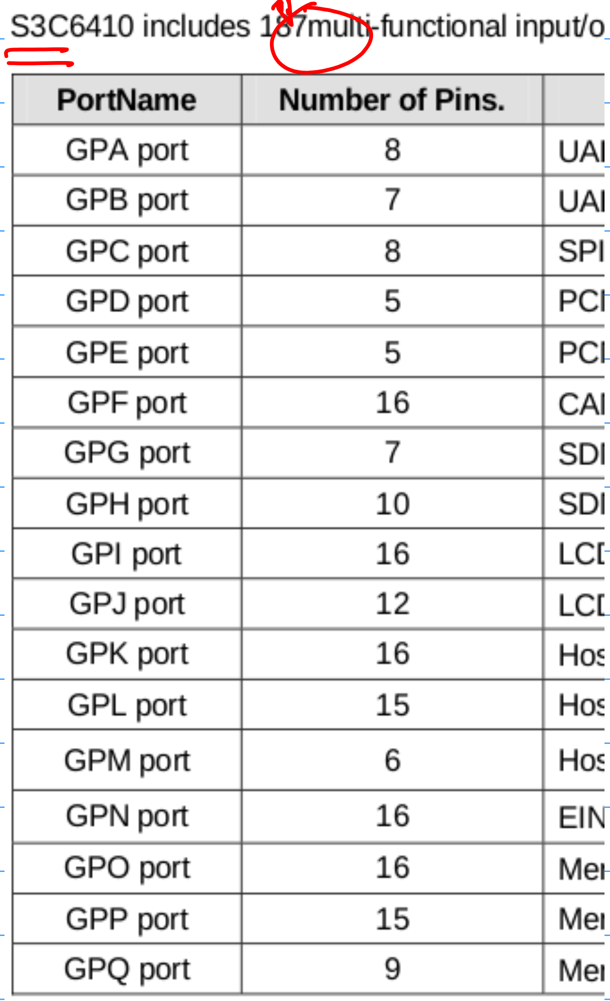
Control Register : Init & Config.
Data Register : Data I/O Operation
Pull up/Down : Electrical Charateristic of the Controller

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

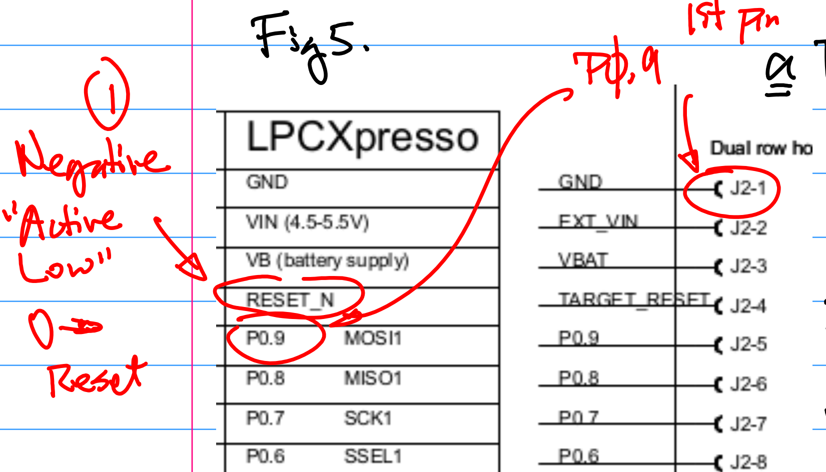
Control Register : GPx CON
GPx : General Purpose Port x
Meaning we have more than one general Purpose Ports.

C. Typical Number of GPP :
LPC1769 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	Me
GPP port	15	Me
GPQ port	9	Me

Fig 5.



a Port (GPP) 0
b Theoretically, pins can be as many as 32

Fig 6.

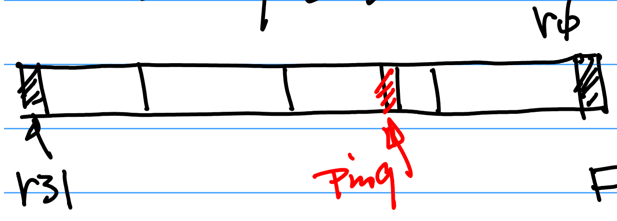


Fig 7

c. GPxDAT Ping:
↑ ,
Port

GPxDAT[9]

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

P0.9 → J2-5 (Connector J2, Pin 5)

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° 2021F-112-Homework (Need Submission to CANVAS)

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

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

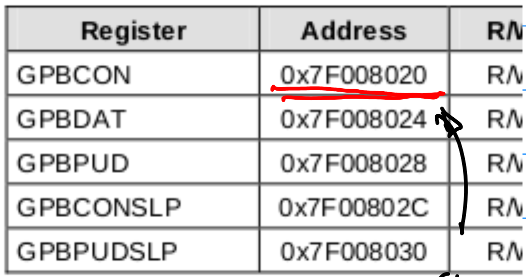
Turn on/off LED via GPIO O/P
Read from (Input) GPIO pin for "1" or "0" when switch is toggled:

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 GPx CON
Number of Possible control functions?

From CPU DataSheet (ARM11), SPRs

①

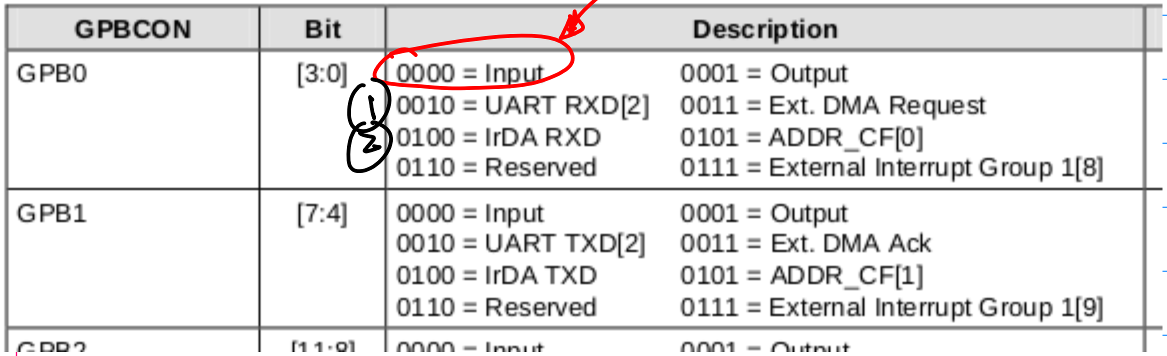


Register	Address	R/
GPBCON	0x7F008020	R/
GPBDAT	0x7F008024	R/
GPBPUD	0x7F008028	R/
GPBCONSLP	0x7F00802C	R/
GPBPUDSLP	0x7F008030	R/

② memory (Address)

CmpE244

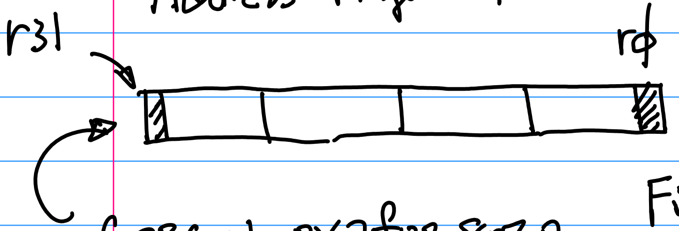
Define GPBϕ as a input pin



GPBCON	Bit	Description	
GPB0	[3:0]	0000 = Input	0001 = Output
		(1) 0010 = UART RXD[2]	0011 = Ext. DMA Request
		(2) 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

Table 1

Note 1: 32 Bit Architecture
↓
32 Bit SPRs (Special Purpose Register)
Address 4 Bytes Apart.



GPBCON 0x7f00_8020

Fig 1.

GPBϕ → pinϕ → CPU, GPBDAT[ϕ]

GPBCON[3:ϕ] = 0000 // Define GPBϕ input

GPBCON[3:ϕ] = 0001 // Define GPBϕ output

GPBCON[3:ϕ] = 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

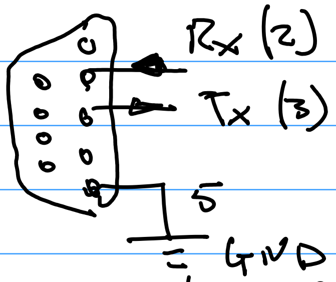


Fig. 2

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;

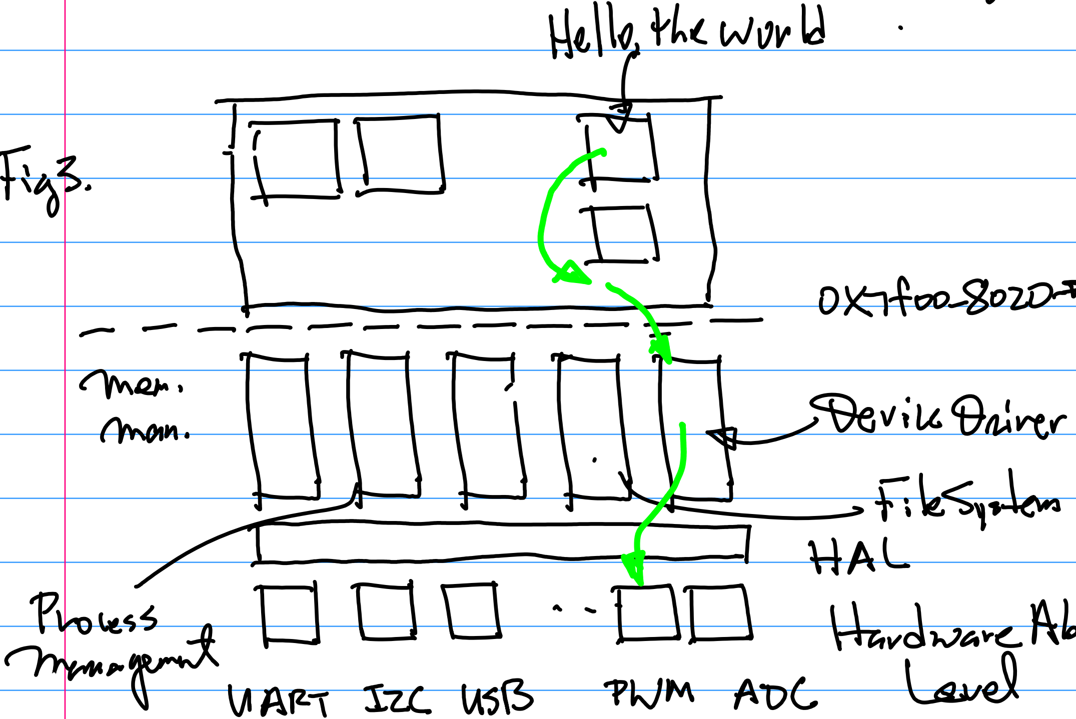
2° Find Driver Program to allow us to config GPP for Input/Output function.

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.

Fig3.



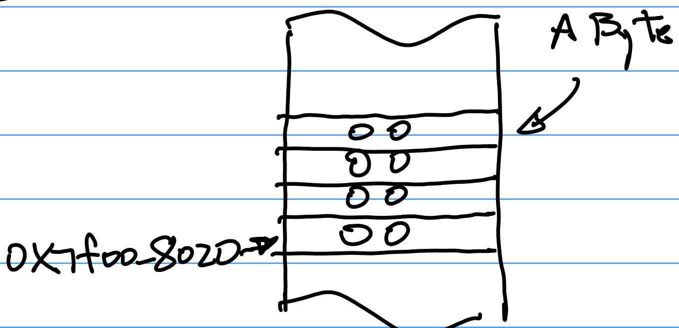


Fig4.

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

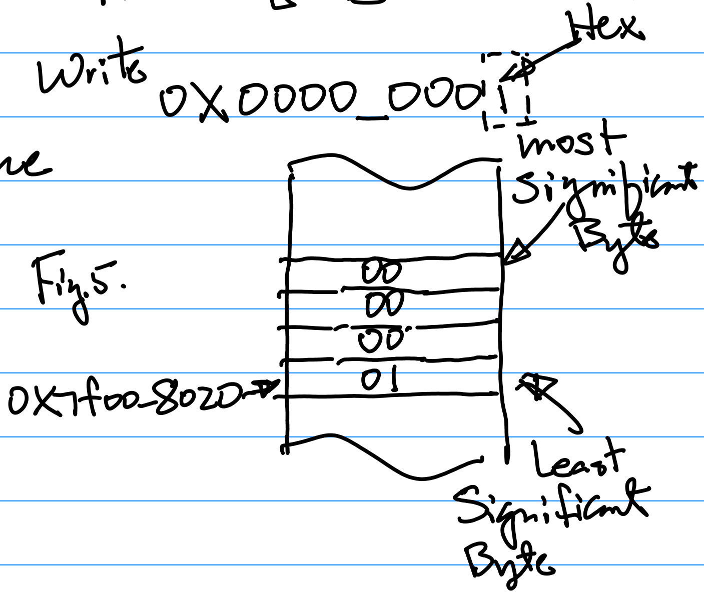
Now, Define GPB0 as an Output, from CPU Datasheet

GPBCON[3:0] = 0001

Write 0X0000_0001

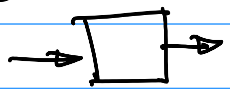
In the kernel space, SPRs, like GPxCON, GPxDAT will fulfill the I/O I/F function.

Fig5.



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 then 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-word.c code

$\underbrace{\text{LPC}}_{\text{Target CPU}}\_\underbrace{\text{GPIO}}_{\text{Peripheral Controller}}\text{->}\ \overline{\text{F}}\text{IO}\overline{\text{D}}\text{IR}$

Fast I/O

Direction

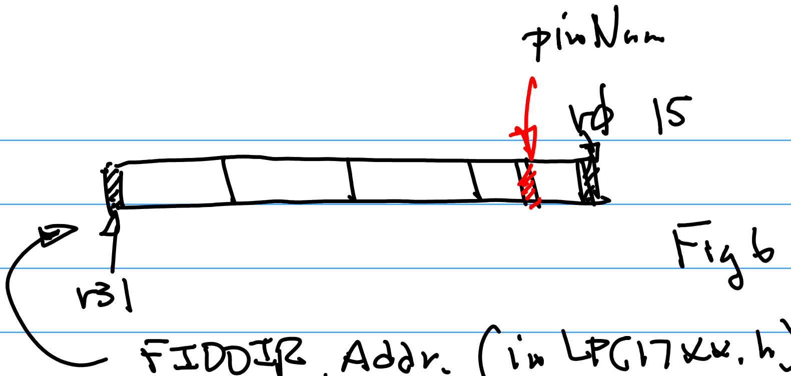


Fig 6

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.

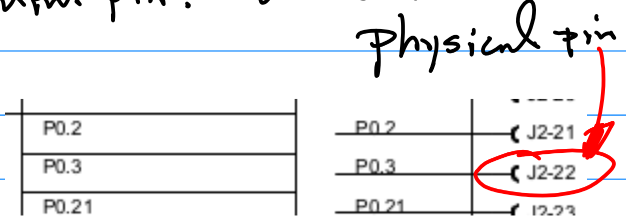


Fig 7.

"|=" bitwise OR

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

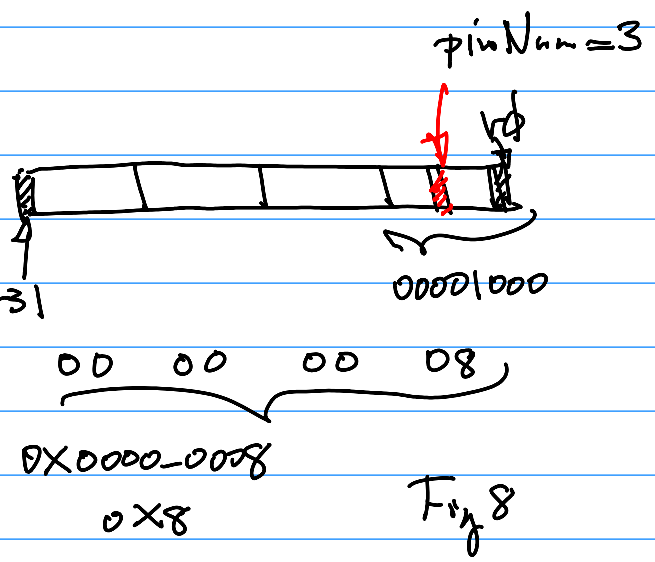
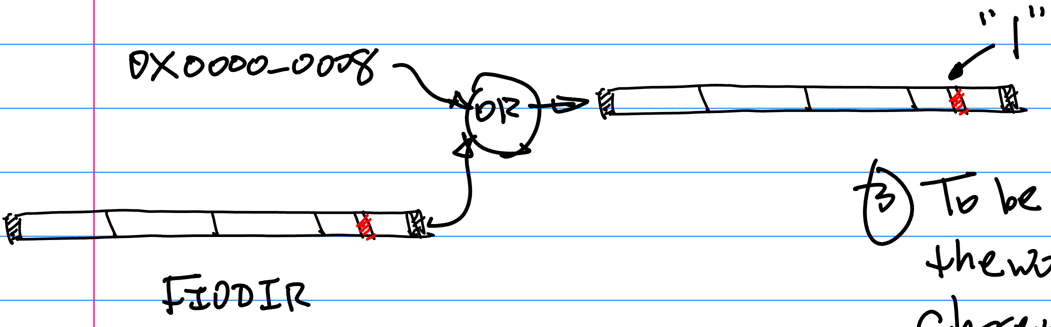


Fig 8



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

LPC_GPIO0 → $\underbrace{\text{FIOSET}}$ Special purpose Register: "DAT" register

Note: ª CPU Datasheet { ARM11 GPP / LPC GPP

ᵇ SPRs { ARM11 GPxCON GPxDAT GPxPUD / LPC FIODIR FIOSET

LPC_GPIO0 → FIOSET = (1 << pinNum);

ᶜ Sample code LPC17xx.h { Peripheral control mem. mapping / GPIO (Last 2~3 pages)

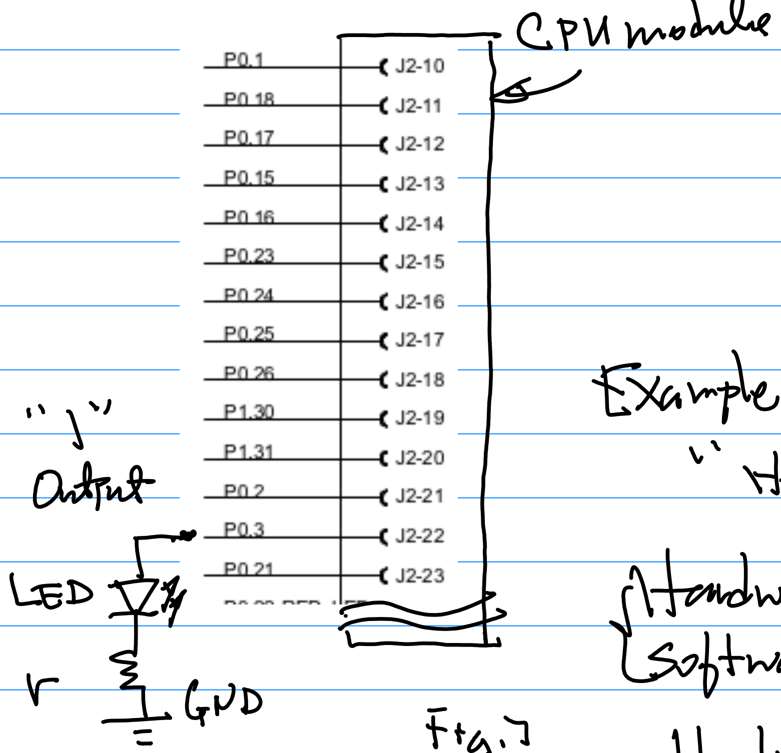


Fig. 7

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

{ Hardware Design / Software Design.

Hardware Design.

Step 1. 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

Note: 1° CMOS [0, 3.3V]

Logic "0" Logic "1"

$I_{CPU} \approx 10mA$

For S/W connected to "A"

Step 2.

$$\text{GPIO}\begin{cases}\text{Input}\begin{cases}\text{"1"}\\\text{"0"}\end{cases}\\\text{Output}\begin{cases}\text{"1" To turn on LED}\\\text{"0" To turn off LED}\end{cases}\end{cases}$$

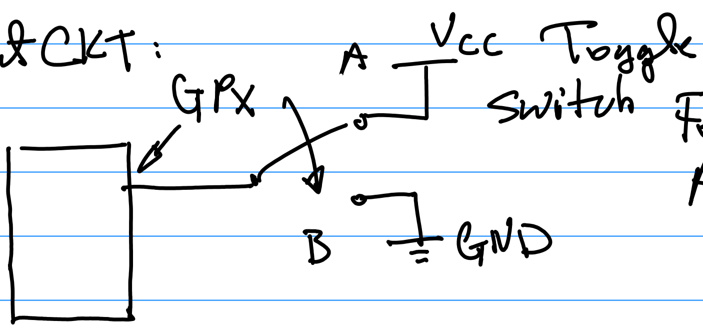
$$\frac{V_{CC}}{R_1} = I_{CPU} \quad \dots (2)$$

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

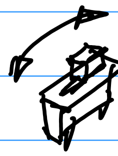
$$\therefore R_1 = V_{CC}/I_{CPU} = \frac{3.3}{10\times10^{-3}}$$

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

Input CKT:



CPU (connector) Fig. 8a



For S/W @ B

Assum GPx is output high

$V_{GPx} = 3.3V$

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

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

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

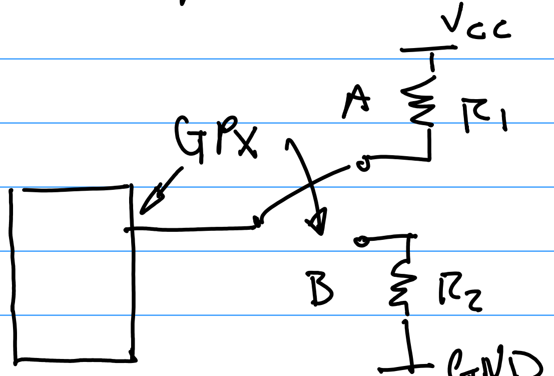


Fig. 8b $R_1 = R_2 \simeq 1K\Omega$

Output Test.

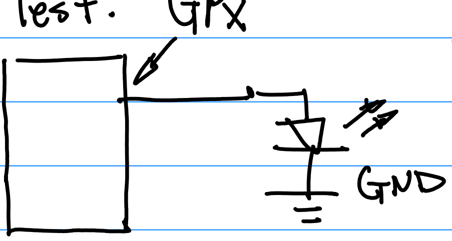


Fig. 9a

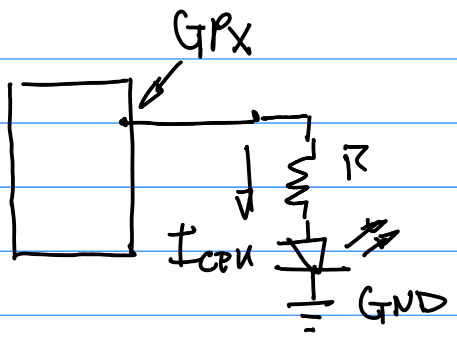


Fig.9b.

Let $I_{CPU} = 10mA$;

$V_{LED} = 1.2VDC$

$$I_{CPU} \cdot R = V_{GPx} - V_{LED} \quad \cdots (4)$$

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

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

$$= 210\,\Omega$$

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