Introduction to Embedded Systems

*Content taken from HP Educational Services, Advanced Embedded Course

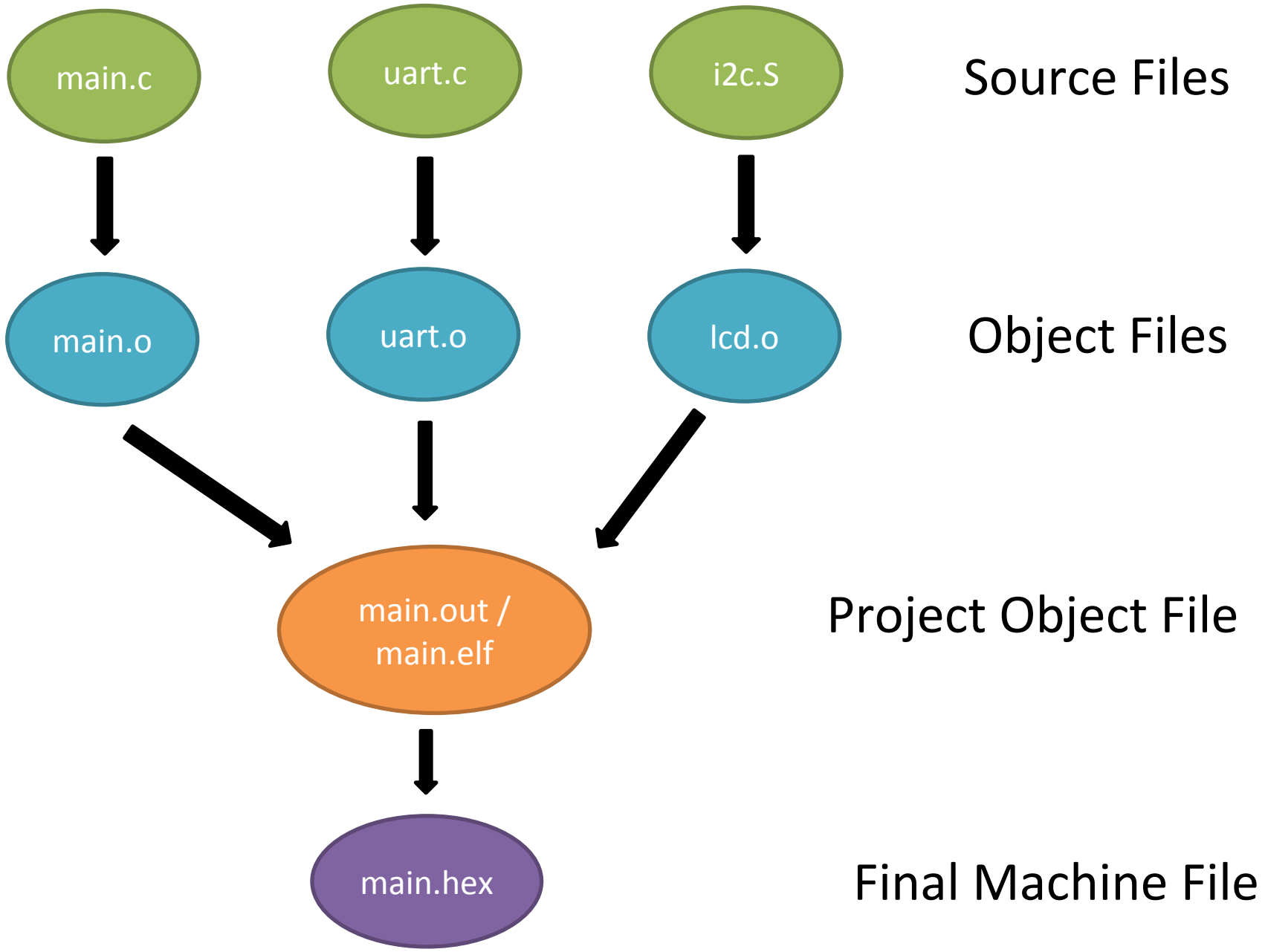
Basics

- What is a microcontroller ? uC vs uP ?
- CMOS-TTL voltage level
- Clock ? RAM ? FLASH ? EEPROM ? Registers ?
- 8 bit / 16 bit / 32 bit ?
- Why AVR ?

Tool-Chain

Bunch of utilities which contains following,

- Compiler .c → .o
- Assembler .S / .asm → .o
- Linker .o → .out / .elf
- Object Translator .elf/.out → .hex
- Programmer .hex → flash of uC
- Debugger For real-time debugging via JTAG
- Simulator For simulating


main.c
uart.c
i2c.S
Source Files
main.o
uart.o
lcd.o
Object Files
main.out /
main.elf
Project Object File
main.hex
Final Machine File

Examples

- AVRGCC / WinAVR
- CVAVR
- Bascom

Using AVRGCC

Install WinAVR on your system.

Open CMD Prompt type : avr-gcc --help

See the manual for more details

Sample Code (main.c)

```
#include <avr/io.h>
// we are not defining which
microcontroller is this !

int main (void){

    DDRA  = 0xFF;
    PORTA = 0xF0;
}
```

1. Compiling

avr-gcc -c -mmcu=atmega128 main.c

This will create main.o

2. Linking

Used when we have multiple source files (.c)

avr-gcc -mmcu=atmega128 main.o -o main.out

This will generate final object code for the project: main.out

3. Object Copy

avr-objcopy -O ihex main.out main.hex

This will generate Final Intel machine file ready to be loaded to Flash of the microcontroller.

4. Programmer (AVR-dude)

Command line programmer

In CMD Prompt type : avrdude for displaying options

avrdude -p atmega128 -P com4 -c jtagmkI -U flash:w:main.hex

Makefiles…

To reduce the number of steps involved in complete process.

Read the Makefile Manual for more details.

First Makefile

Create a file in the project folder WITHOUT any extension with following name : Makefile

Write Following code

```
main:
    avr-gcc main.c -mmcu=atmega128 -c
    avr-gcc -mmcu=atmega128 main.o -o main.out
    avr-objcopy -O ihex main.out main.hex

program:
   avrdude -p atmega128 -P com4 -c jtagmkI -U flash:w:
main.hex
```

Using Makefile

Type following on CMD Prompt:

make main : will execute commands on label ‘main’

make program : will execute commands on label ‘program’

Completing Makefile

```
PROJ = main
# Name of project without extension
SRC = main.c lcd.c uart.c
# List all Source Files here.
CC = avr-gcc
#Compiler used
MCU = atmega128
#Microcontroller used

main:
	$(CC) $(SRC) -mmcu=$(MCU) -c
	$(CC) -mmcu=$(MCU) *.o -o $(PROJ).out
	avr-objcopy -O ihex $(PROJ).out $(PROJ).hex

program:
	avrdude -p $(MCU) -P com4 -c jtagmkI -U flash:w:$(PROJ).hex

clean:
	rm *.o *.out *.hex
```

Auto Generated Makefile

Use Mfile generator from WinAVR. Edit following lines,

- Line 44 : MCU = atmega128
- Line 65 : F_CPU = 8000000
- Line 73 : TARGET = main
- Line 83 : SRC = $(TARGET).c uart.c lcd.c
- Line 97 : ASRC = i2cmaster.S
- Line 276 : AVRDUDE_PROGRAMMER = jtagmkI
- Line 279 : AVRDUDE_PORT = com4

IDE : Integrated Development Environment

- AVR Studio : backend AVRGCC

All tools are inbuilt.

C Review

Macros

#define Label Replacement

Function through Macros

#define add(x,y) (x)+(y)

Binary Operators

Operation	Operator Symbol in C	Operation Mode
AND	&	Binary
OR	|	Binary
XOR	^	Binary
1's Complement	~	Unary
Left Shift	<<	Binary
Right Shift	>>	Binary

Shift Operators

Right Shift : >>

Left Shift : <<

Bit operations on Registers

Setting a bit

Use the bitwise OR operator (|) to set a bit.

REGA |= 1 << x

Clearing a bit

Use the bitwise AND operator (&) to clear a bit.

REGA &= ~(1 << x)

Bit operations on Registers

Toggling a bit

The XOR operator (^) can be used to toggle a bit.

```
REGA ^= 1 << x;
```

Checking/Reading a bit

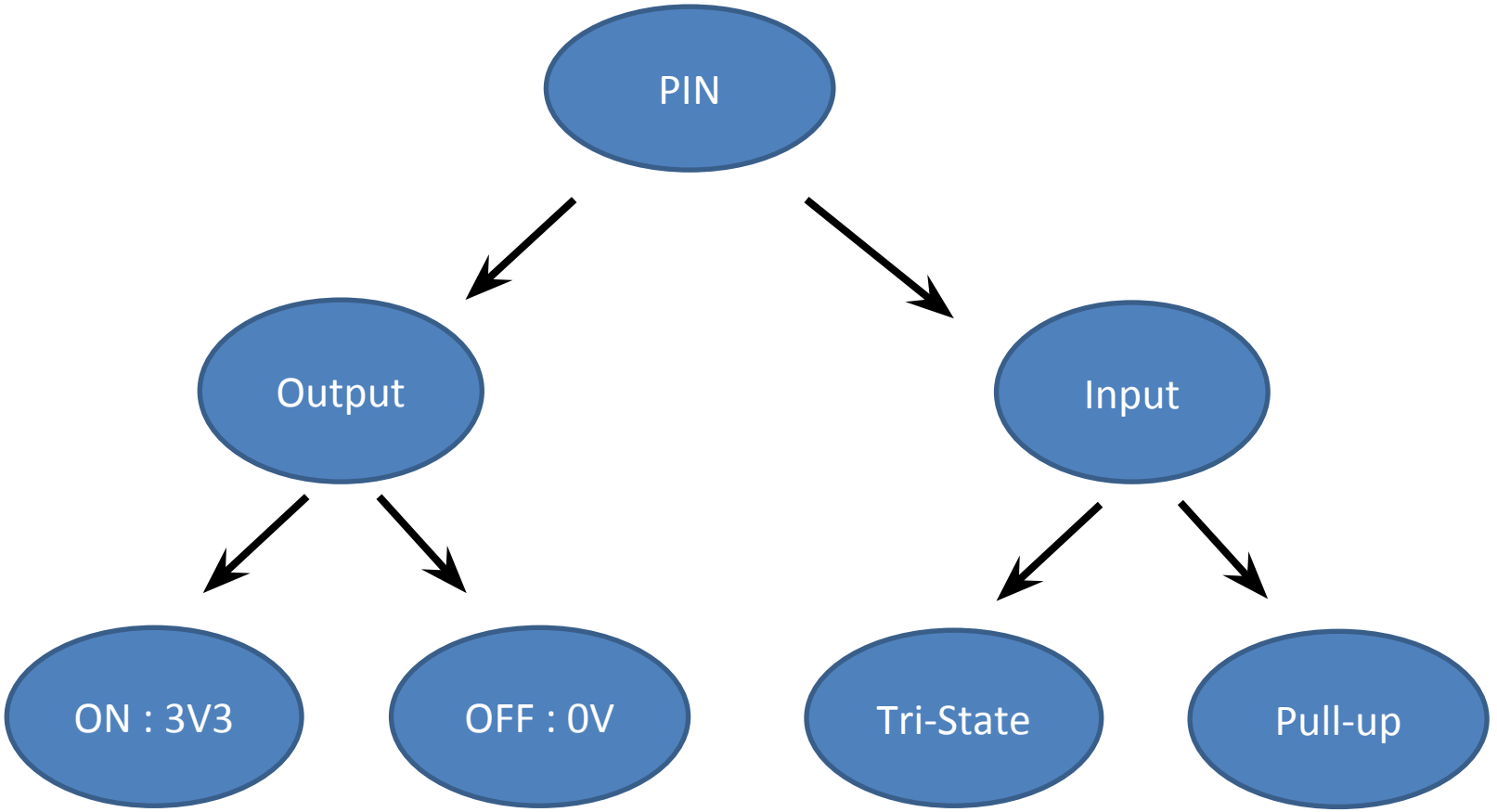
```
bit = REGA & (1 << x);
```

AVR GPIO

3 Registers

8 bit registers mapped to GPIO pins.

- PORTX
- PINX
- DDRX



DDRA

PORT A

PINA

Port Registers

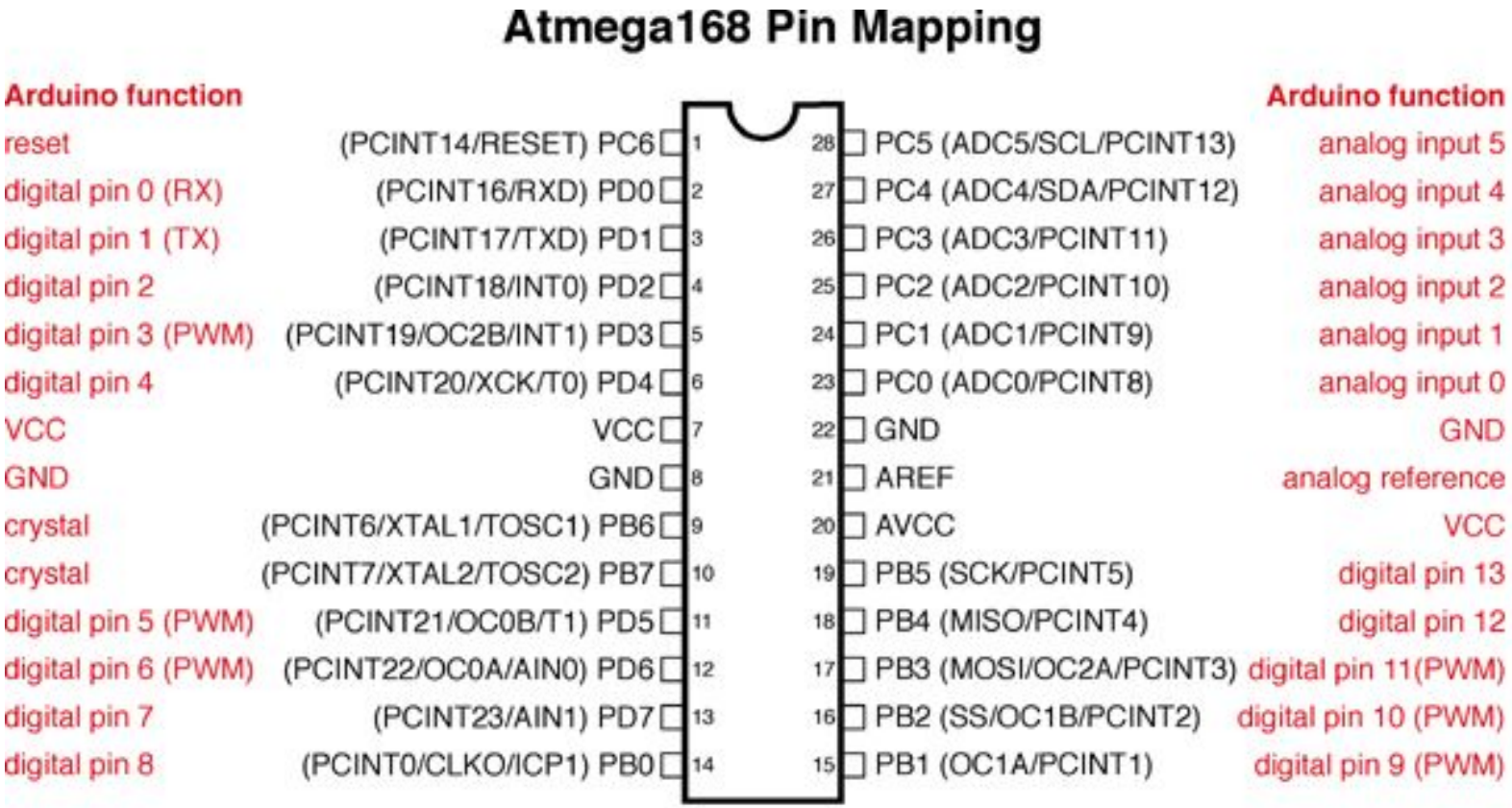
Port registers allow for lower-level and faster manipulation of the i/o pins of the microcontroller on an Arduino board. The chips used on the Arduino board (the ATmega8 and ATmega168) have three ports:

- B (digital pin 8 to 13)
- C (analog input pins)
- D (digital pins 0 to 7)

Port Registers Conti...

- **PORTD** maps to Arduino digital pins 0 to 7
- **PORTB** maps to Arduino digital pins 8 to 13 The two high bits (6 & 7) map to the crystal pins and are not usable
- **PORTC** maps to Arduino analog pins 0 to 5. Pins 6 & 7 are only accessible on the Arduino Mini

ATmega168/328-Arduino Pin Mapping



Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Presenter: Varun Kumar (B.Tech IIITD)

Author: Sourabh Sankule,
Team Lead, Sandisk India
(B.Tech IITK 2009)

Cyborg FB page : http://tinyurl.com/cyborg-fb

Cyborg website : http://tinyurl.com/cyborg-web

Electroholics FB page : http://tinyurl.com/electroholics-fb

Electroholics Website : http://tinyurl.com/electroholics

Contact us

Varun Kumar

Email ID :
varun13169@iiitd.ac.in

Phone number :
09716057062

Diljyot Singh

Email ID :
diljyot13132@iiitd.ac.in

Phone number :
08527055549