

Introduction to Embedded System Design

Lecture - 1: Course Coverage. Preliminaries. Terminology.

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

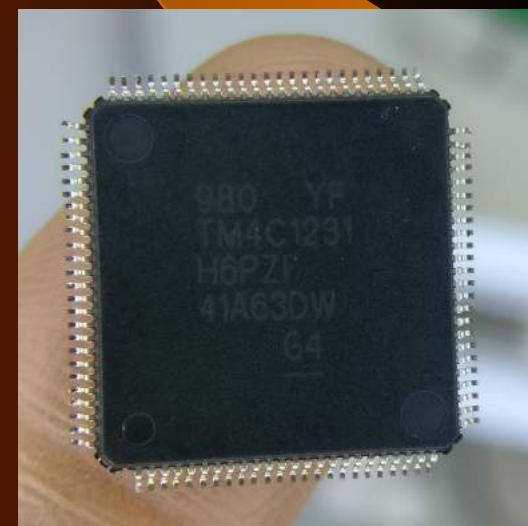
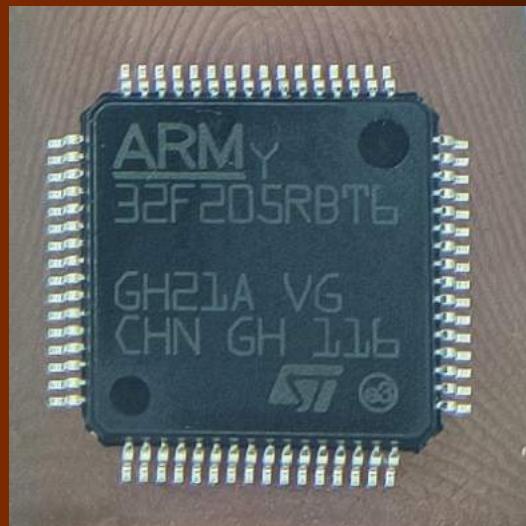
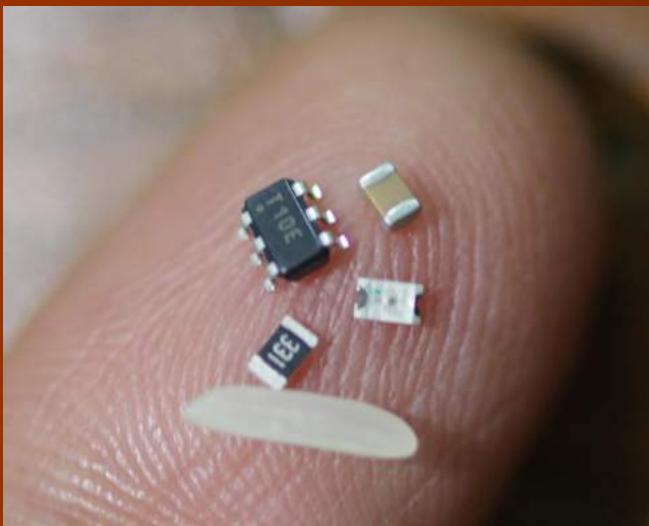
Indian Institute of Technology,
Jammu

Course Objectives - 1

- Learn about the elements of embedded systems in general and their applications.
- Learn embedded system design using a building block approach. These blocks cover input devices and sensors, output devices and actuators, communication links, storage devices, power supply and of course the embedded computer.
- Learn various ways of implementing the embedded computer block, specially a Microcontroller.
Microcontrollers are complete computers on a single chip.

Course Objectives - 2

- These microcontrollers have great diversity in terms of size and in terms of performance and we will give you some idea about that.



Course Objectives - 3

- Learn about TI's MSP430 - architecture, programming and interfacing. We will teach you Embedded C programming and how to write, debug and download the compiled program into the memory of the microcontroller.
- You will also learn about aspects of complete system design, including testing and debugging.
- At the end of this course, you can expect to be able to design simple embedded systems from scratch to finish.

Course Objectives - 4

- But larger motive is for us to make you fall in love with electronics.
- To enthuse you to build circuits and systems - From simple circuits to more complex ones and eventually, to be able to visualize and build complete systems.

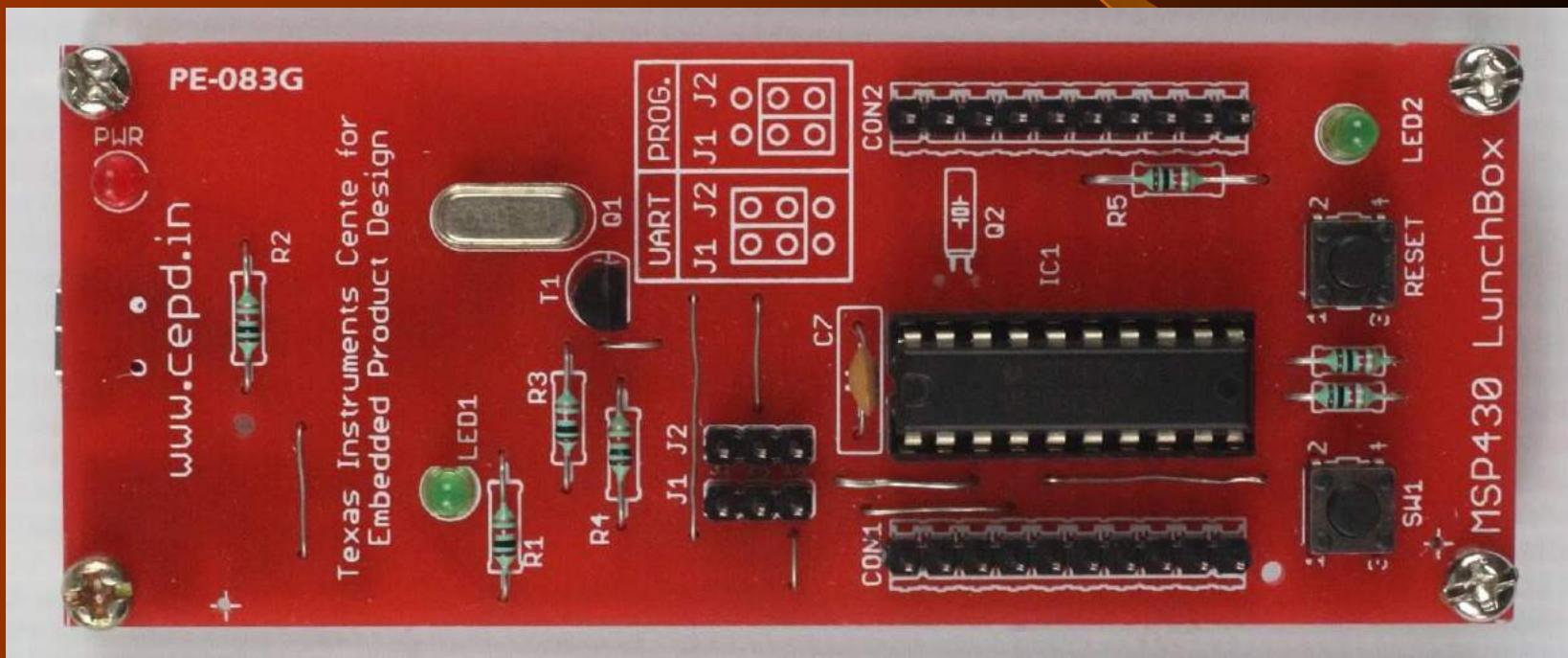
Prerequisites for the Course

We expect you to know

- How basic electronic components and circuits work
- Elements of digital circuits and systems, including an idea about Finite State Machines.
- Some experience in C programming
- Idea of computer architecture

Logistics - 1

MSP430 LunchBox Microcontroller Evaluation Kit



<http://dvgadre.blogspot.com/2017/01/make-yourself-msp430-lunchbox-for-1.html?m=1>

Logistics - 2

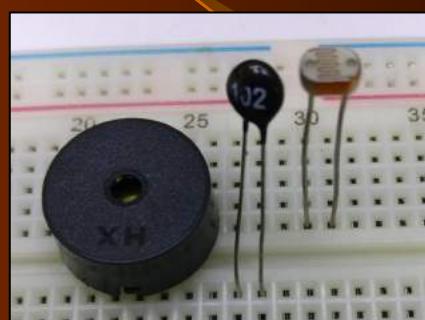
Components Kit



Switches



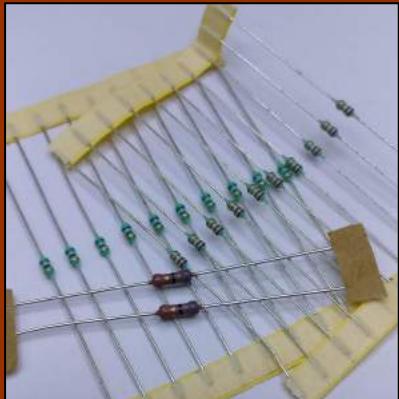
ICs: ULN2003
and 555



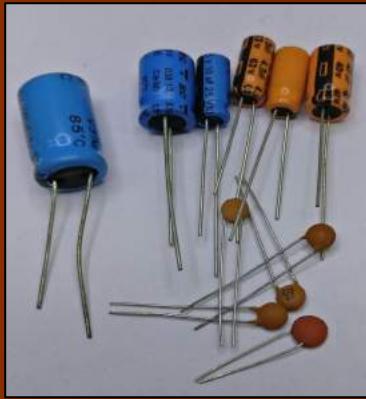
Buzzer, NTC
and LDR



Preset



Resistors



Capacitors



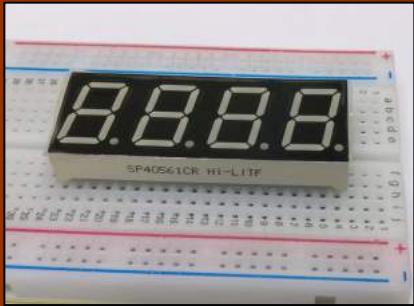
Potentiometer



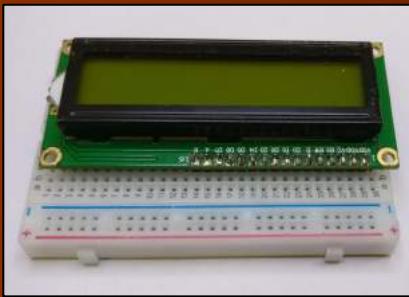
LEDs

Logistics - 2

Components Kit



Seven Segment
Display



16 x 2 LCD



Breadboard



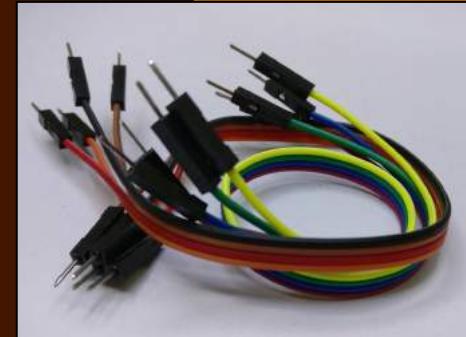
Hookup wires

- You should also be familiar with Fritzing and be able to use it:

<https://fritzing.org/home/>

- You should also be familiar with Eagle CAD:

<https://www.autodesk.in/products/eagle/overview>



Jumper Wires

Definitions and characteristics of an Embedded System

- *A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function.*
- *Any computer system whose end objective is not primarily computational.*

Definitions and characteristics of an Embedded System

- *Any computer system other than the traditional desktop or laptop system. Lately, the smartphone also falls in the category of general purpose computing device!*
- *In some cases, embedded systems are part of a larger system or product, as is the case of an airbag deployment system or anti-lock braking system in a car.*

Ubiquity of Embedded Systems

- Work place: Printers, scanners, network switches
- Banks: ATM, Passbook printer.
- Hospitals: Medical Equipment
- Industry: Industrial equipment, automation.
- Agriculture: Drip irrigation, soil quality instrument.
- Supermarkets: POS, scanner.
- Defence and Space: Missiles, rockets, satellites, space probes.
- Transport: locomotives, cars.
- Telecom: Mobile broadband equipment, switches, Telephone exchanges.
- Entertainment: Projection systems in Cinema halls, 4D seats

Embedded System Application Areas

- Small and single microcontroller applications: small toys, home gadgets etc.
- Control and automotive systems: ABS, Cruise control etc.
- Distributed Embedded Control: Networked Industrial control applications, automotive.
- Networking: Network switches, routers.
- Critical systems: Nuclear, medical and aviation devices.
- Robotics: Warehouse robots, assembly line robots.
- Computer peripheral applications: Portable HDD, Printers, scanners.
- Signal processing: Radar, Security cameras.

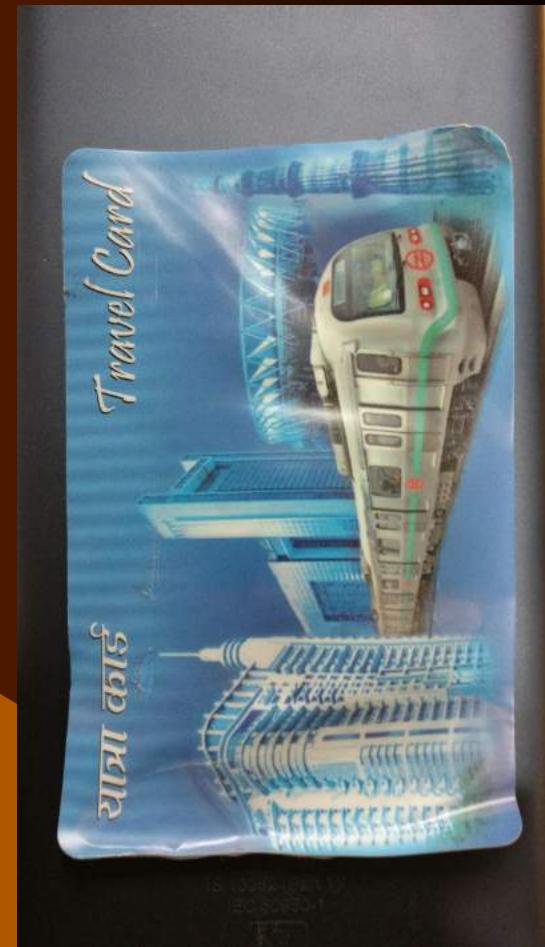
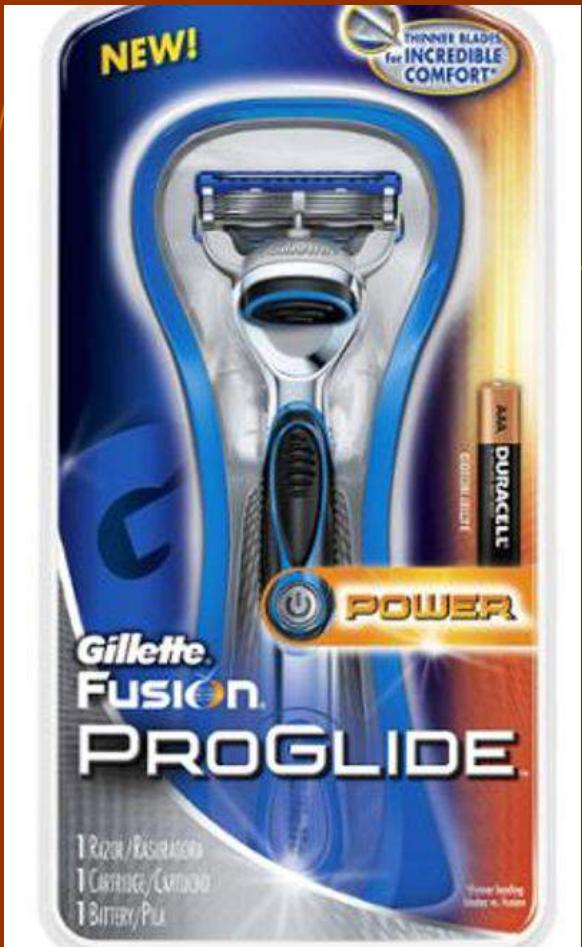
Examples of Embedded Systems at Home

- Communication: Mobile phone, Landline phone, Modem.
- Entertainment: TV, TV Remote!, Set top box, Music system, Noise cancelling headphone, digital picture frame.
- Convenience: Washing machine, RO Water Purifier, Microwave oven, Shaving razors (certain kinds).

Examples of Embedded Systems at Home - 2

- Comfort: Air conditioner, fancy hot water geyser, Mood lamp.
- Health: Treadmill, blood sugar and pressure measurement, Fitness Tracker watches.
- Utilities: Electricity (electronic) meter, RFID tags.
- Transport: Car, Scooter/Motorcycle, Electric bicycle.

Unbelievable Examples



<https://www.element14.com/community/thread/15217/lis-msp430c092-mixed-signal-microcontroller-in-gillette-razors?displayFullThread=true>

Embedded Computers in Home Gadgets? Why?

Let us consider two examples:

1. air conditioner
2. washing machine

Air Conditioner

- A Compressor using an AC motor, coolant and heat exchanger
- Mode Selector Switch to select cooling: Low, Medium, High
- Relay/Contactor
- Thermostat (Bi-metallic strip type)

Air Conditioner with Embedded Computer

- A Compressor using an AC motor, coolant and heat exchanger
- Digital Display of Temperature: Set-point and ambient
- Remote Control Operation
- Relay/Contactor
- Semiconductor/Thermocouple based temperature sensor

Washing Machine

- Outer Enclosure, middle container and inner tumbler.
- Two pipes in middle container: Water inlet and outlet with valves.
- Inner tumbler with holes to let water fill.
- Selector switch with time setting

Washing Machine with Embedded Computer

- Two pipes in middle container: Water inlet and outlet with valves.
- Inner tumbler with holes to let water fill and agitator.
- Selector switch with various settings for clothes type, wash cycle.
- Digital Display for modes, time for wash cycle
- Water level sensor, dirt sensor etc.
- Motor direction control for agitation.
- Ability to resume wash cycle after power failure, audible alerts etc.

Embedded System: Some Observations

- Embedded Systems is a big, fast growing industry (For India alone, US\$ 500 billions by 2020; $500*1000*7$ Crore Rupees = 35 Lakh Crore Rupees.)
- Microcontrollers form quite common core for embedded systems.
- The Software running on this core makes the embedded system tick...

Related Fields

- Physical Computing
- Cyber Physical System
- Internet of Things (IoT)
- Embedded Systems

Comparing Embedded System (ES) and General Purpose (GP) Computing Systems

- ES are dedicated to specific tasks.
- ES can be implemented using wide variety of processors, even generic or custom.
- ES are cost sensitive.
- ES operate under real time constraints

ES Vs. General Purpose Computing Systems

- ES are often designed to operate in extreme environmental conditions.
- ES usually run out of ROM.
- ES have resource constraints.
- ES are infrequently reprogrammed.
- ES have hard reliability and correctness requirements.

Demystifying Terminology

- Computer: CPU, Memory and I/O Ports.
- Microprocessor: CPU on a single chip
- Microcomputer: Microprocessor + Memory + I/O on a single PCB
- Microcontroller: Microcomputer on a single chip
- System on Chip (SoC): Microcontroller + programmable analog!

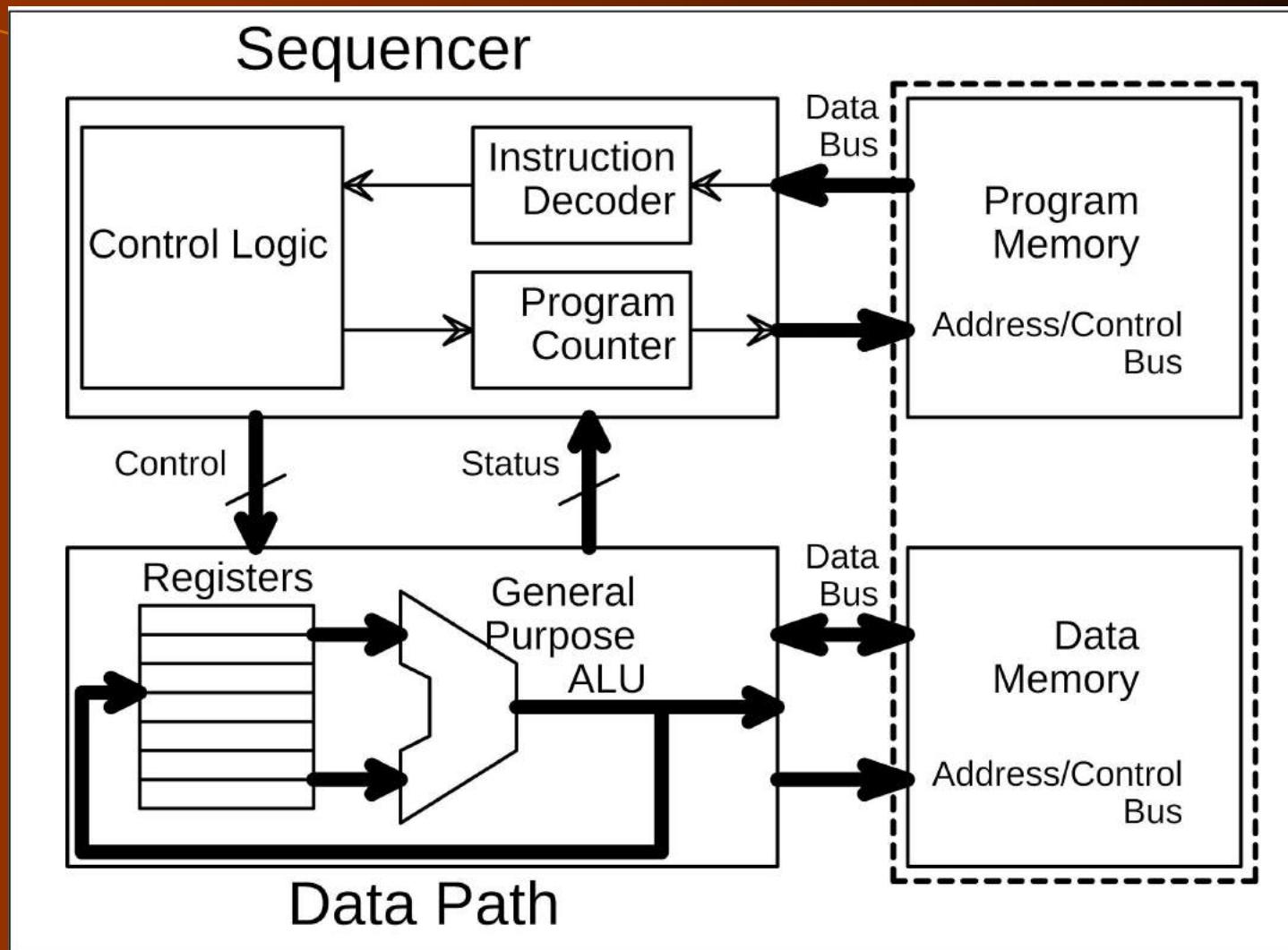
System on Chip

- Microcontroller + programmable Analog subsystem on a single substrate
- General context:
Microprocessor/Microcontroller + custom functional units on a single substrate

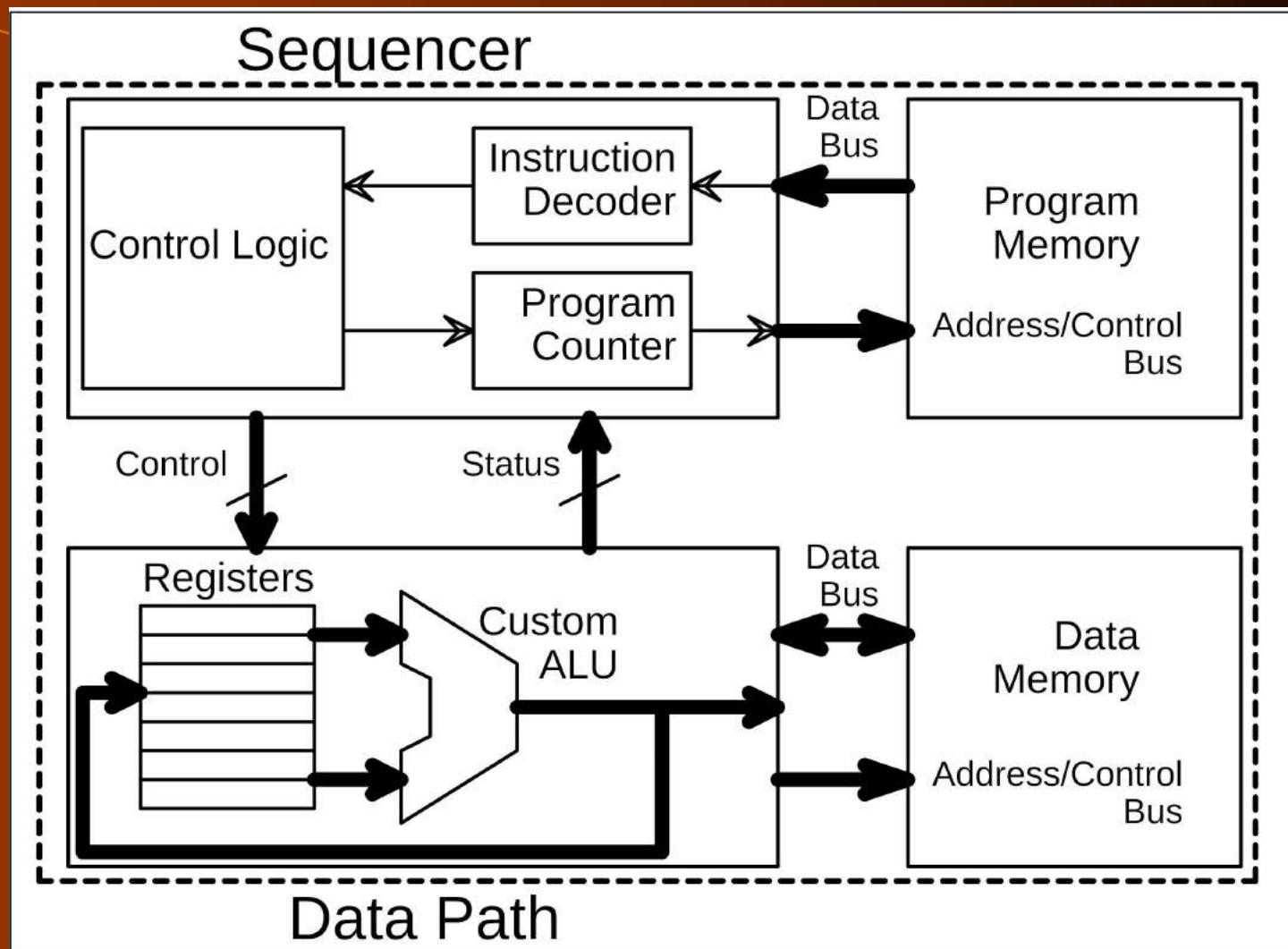
Implementing the Embedded Computer

- General Purpose Processor (GPP)
- Application Specific Processor (e.g.
Microcontroller, Digital Signal Processor)
- Single Purpose Computer

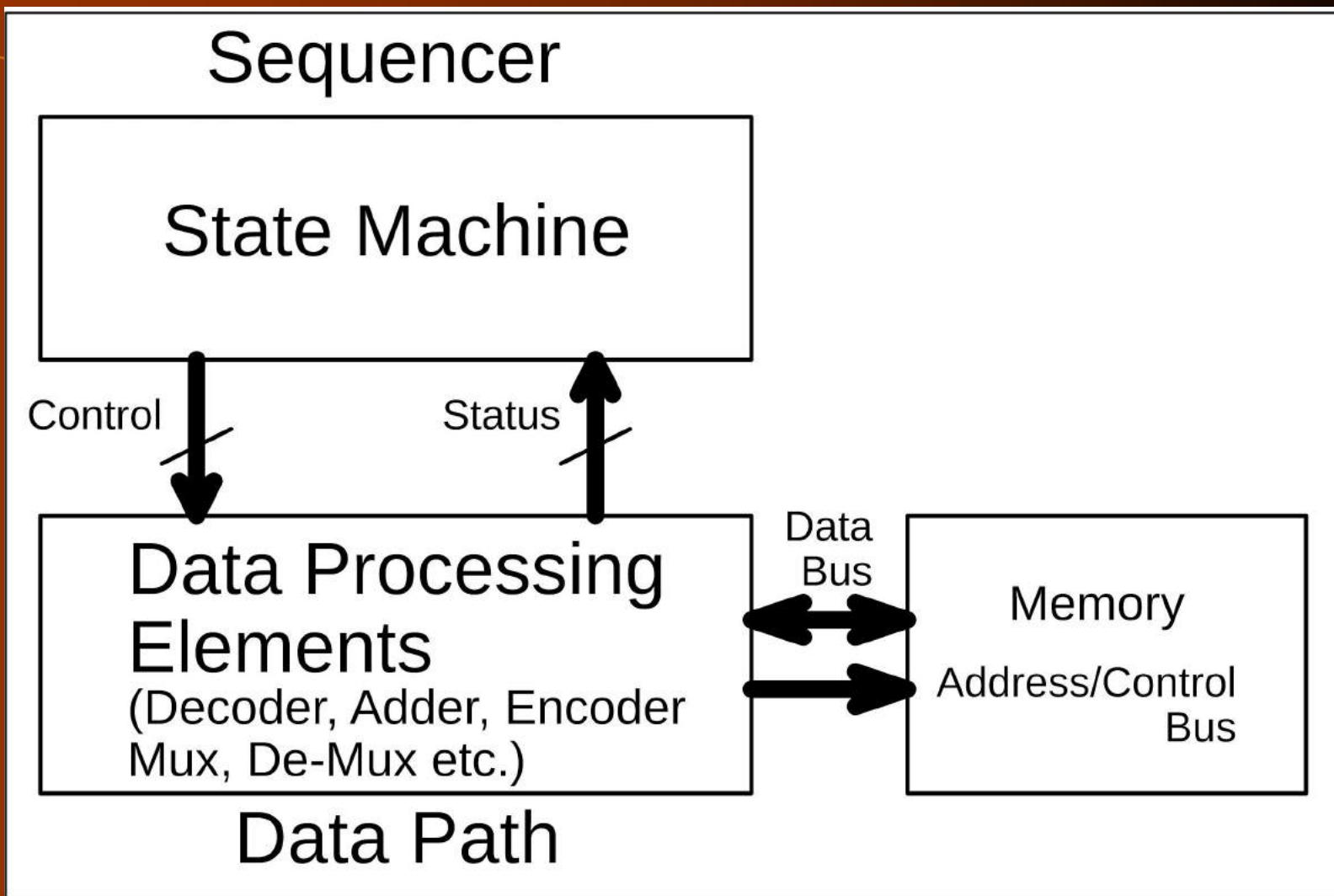
General Purpose Processor



Application Specific Processor

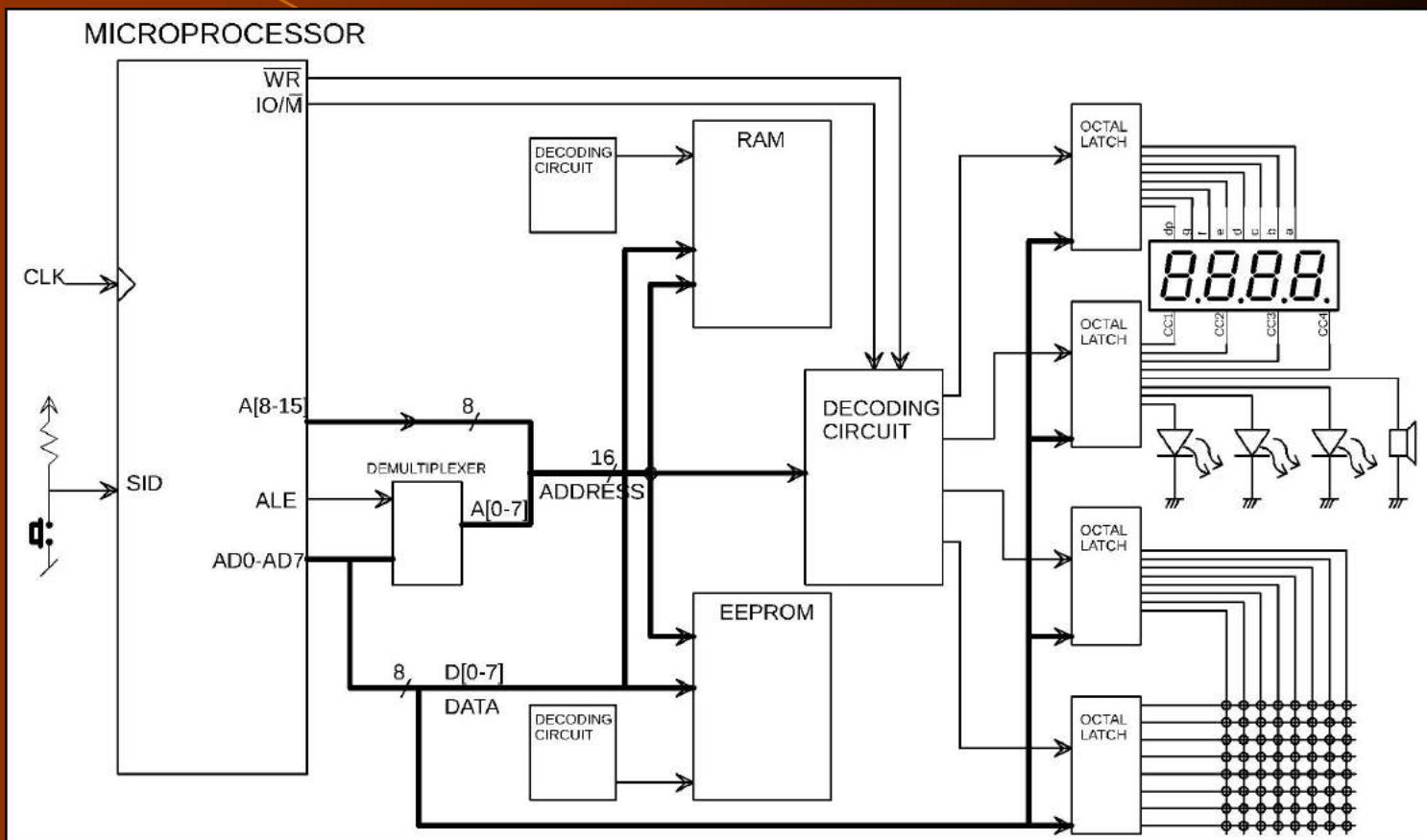


Single Purpose Processor



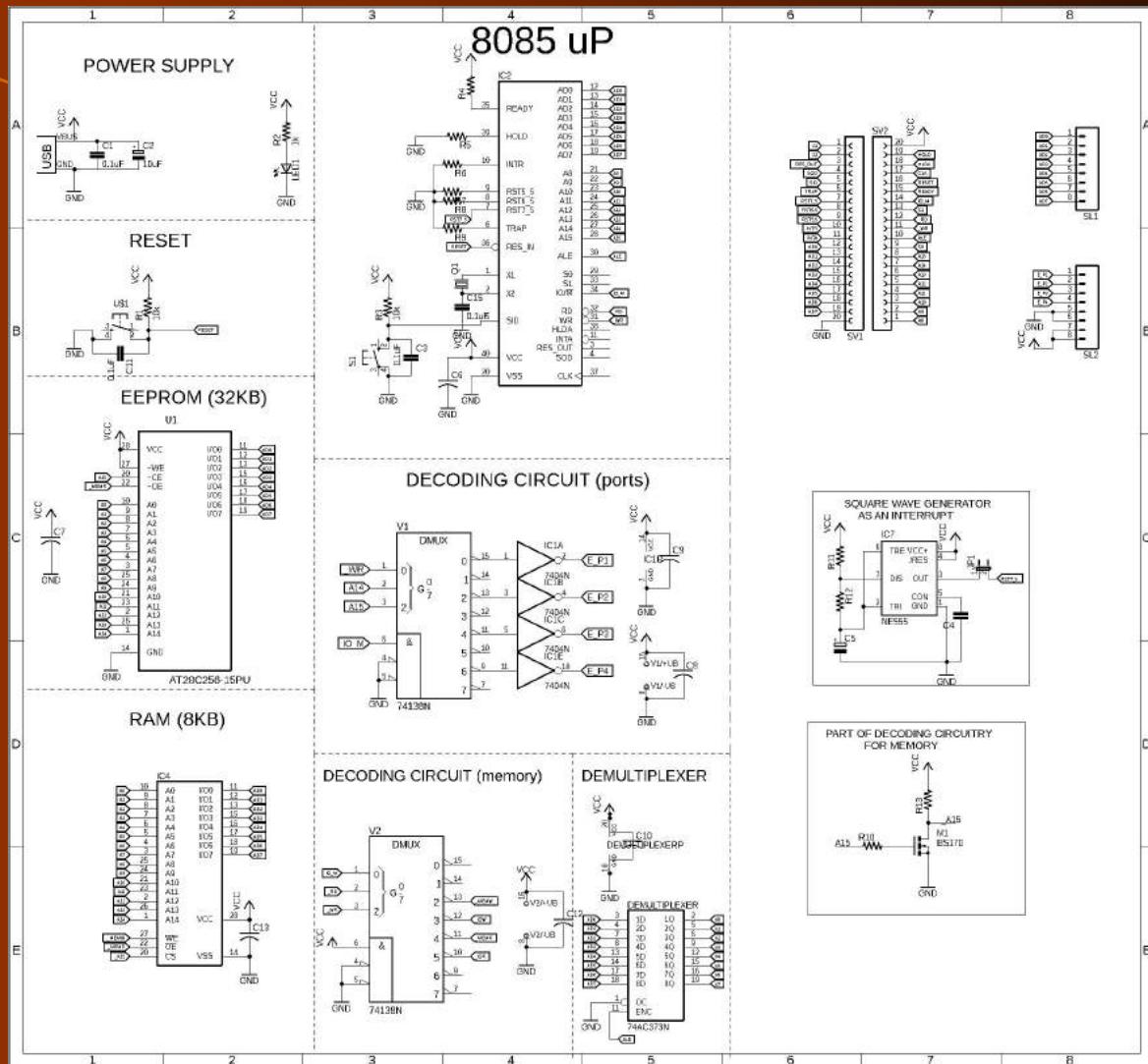
General Purpose Processor Based Embedded Computer

Block diagram of 8085 Based Roulette

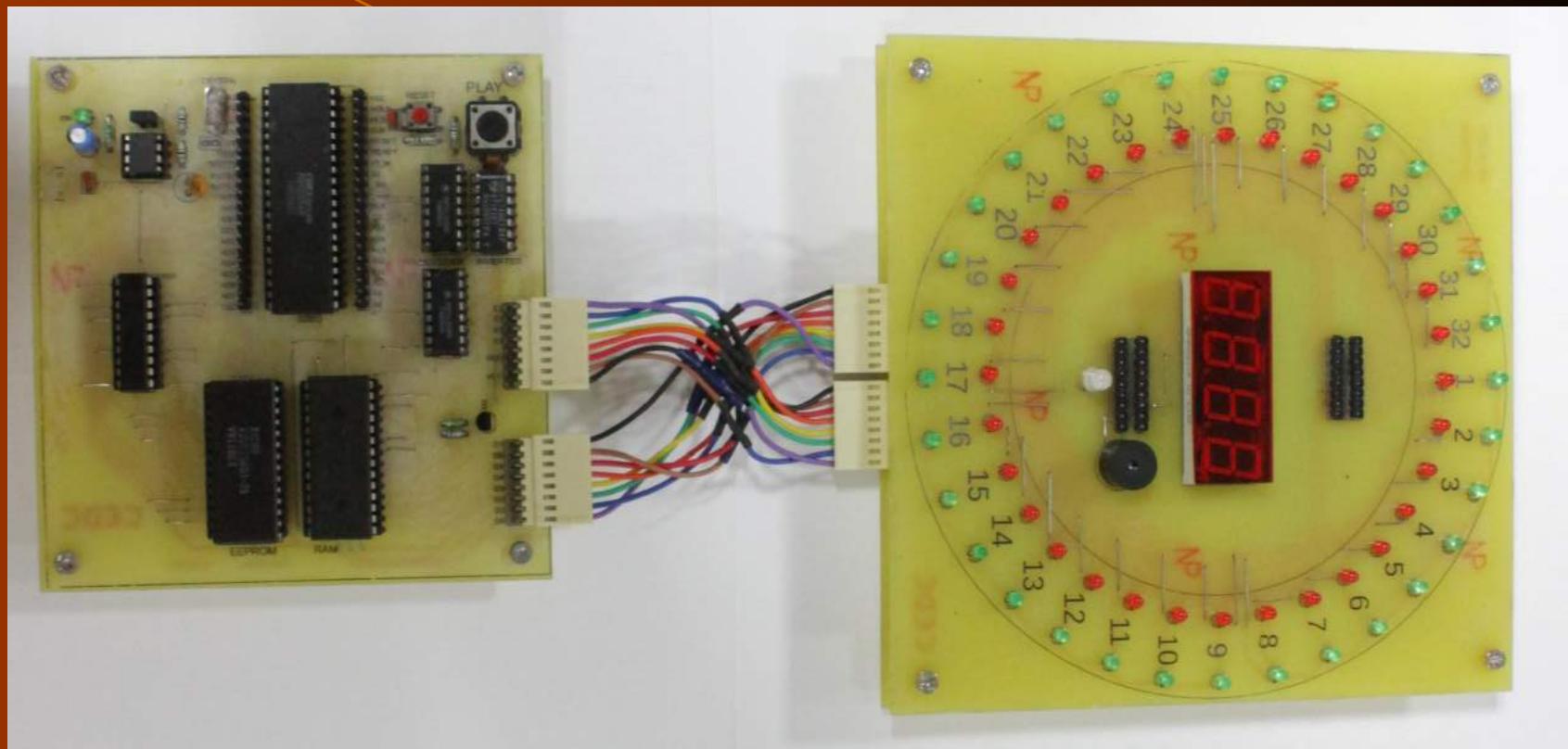


General Purpose Processor Based Embedded Computer

Schematic diagram of 8085 Based Roulette

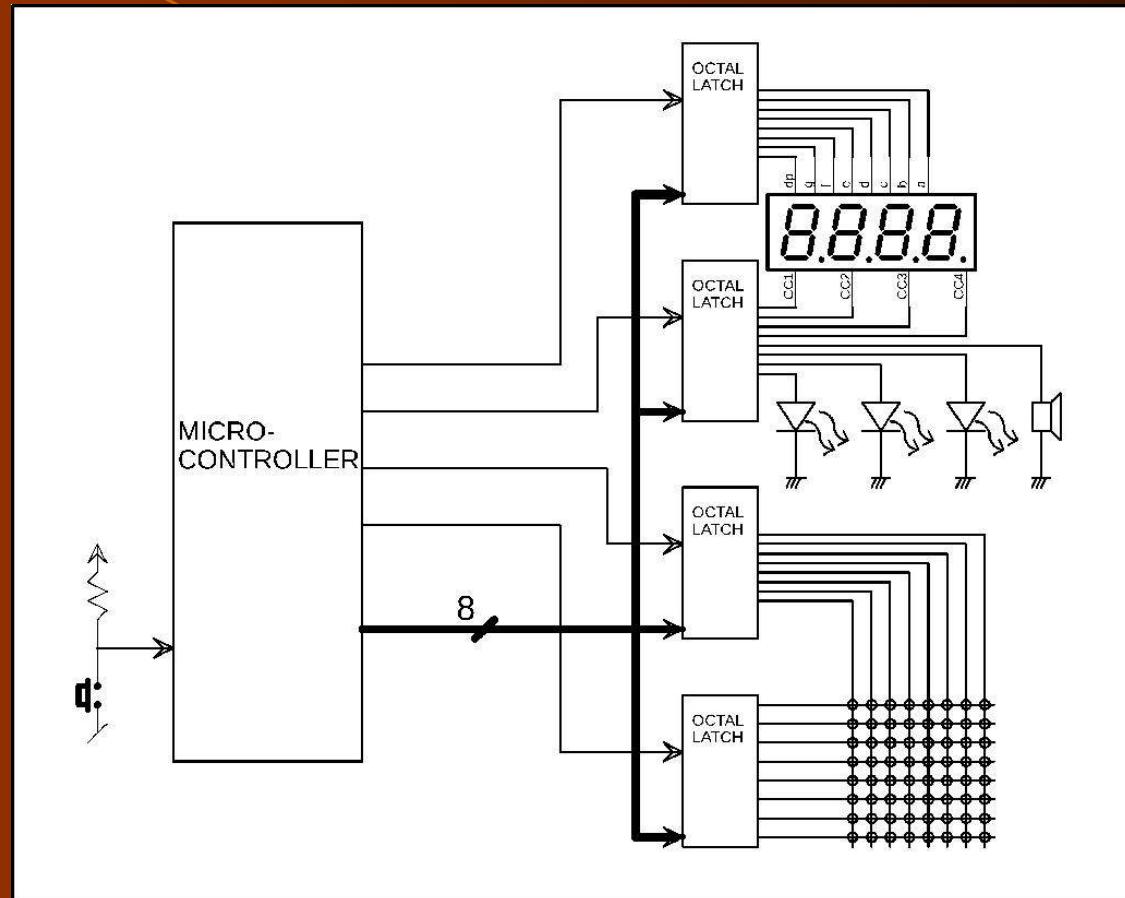


Photograph of the 8085 based Roulette Wheel Game



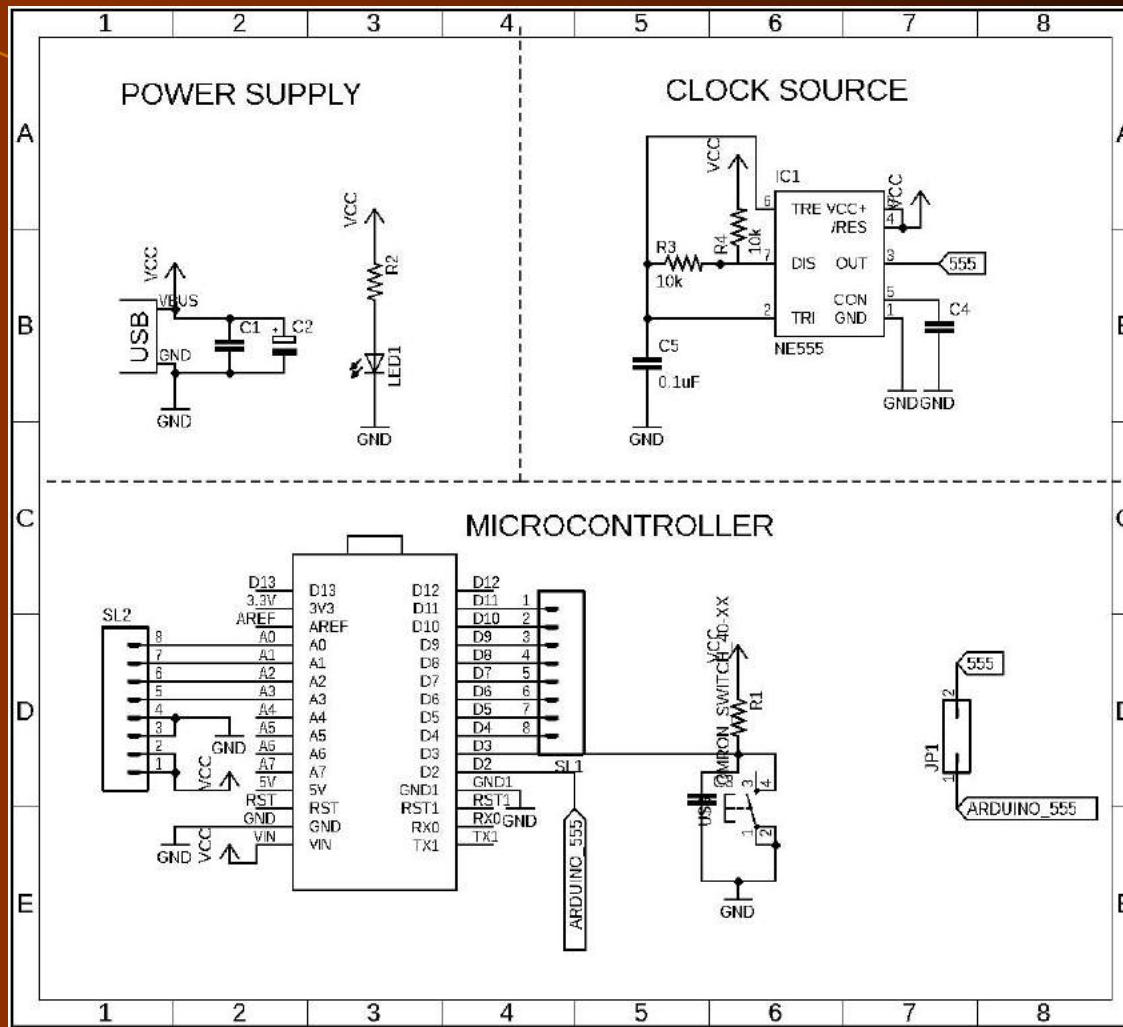
Application Specific Processor Based Embedded Computer

Block diagram of Arduino Based Roulette

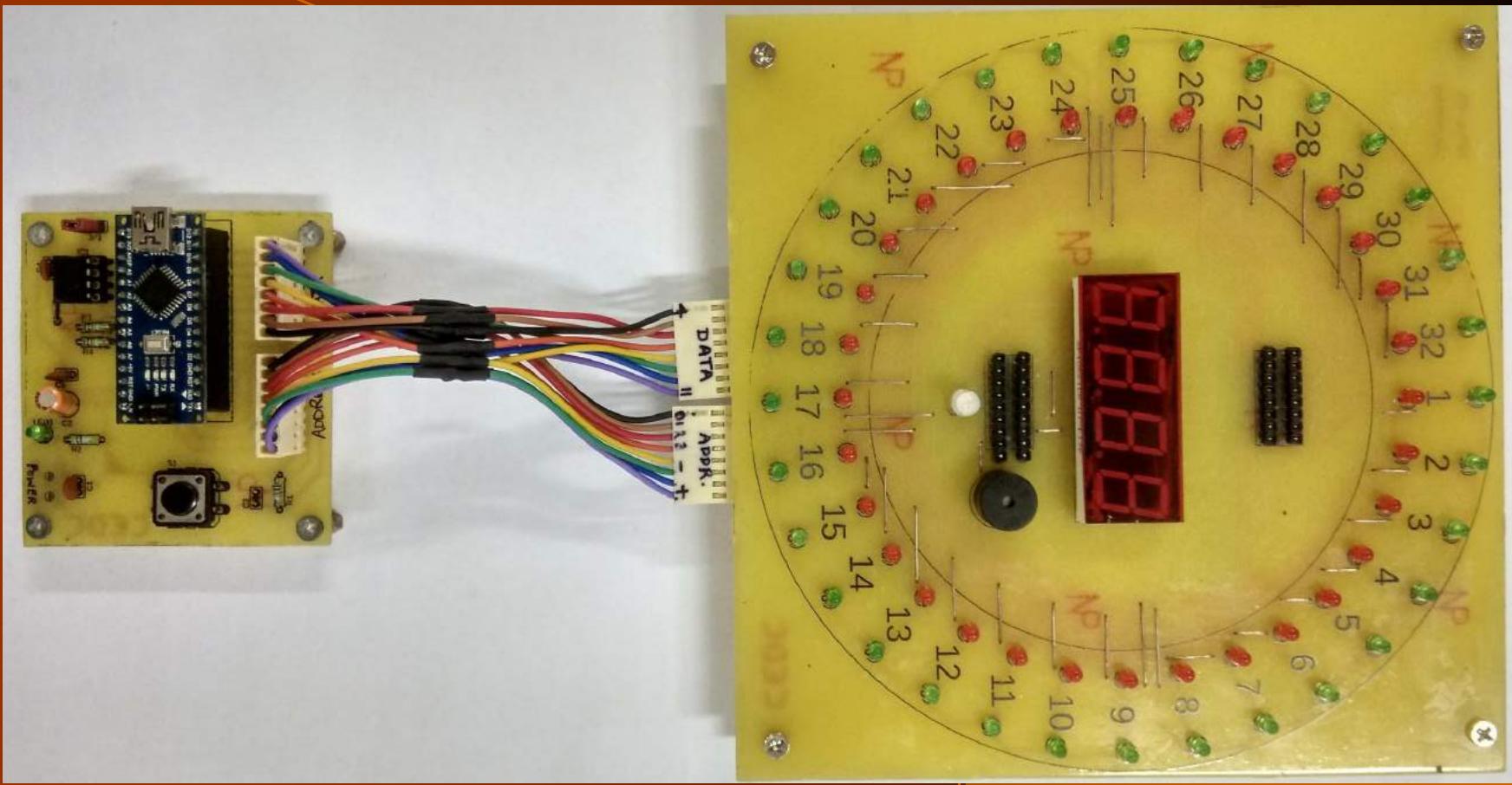


Application Specific Processor Based Embedded Computer

Schematic diagram of Arduino Based Roulette

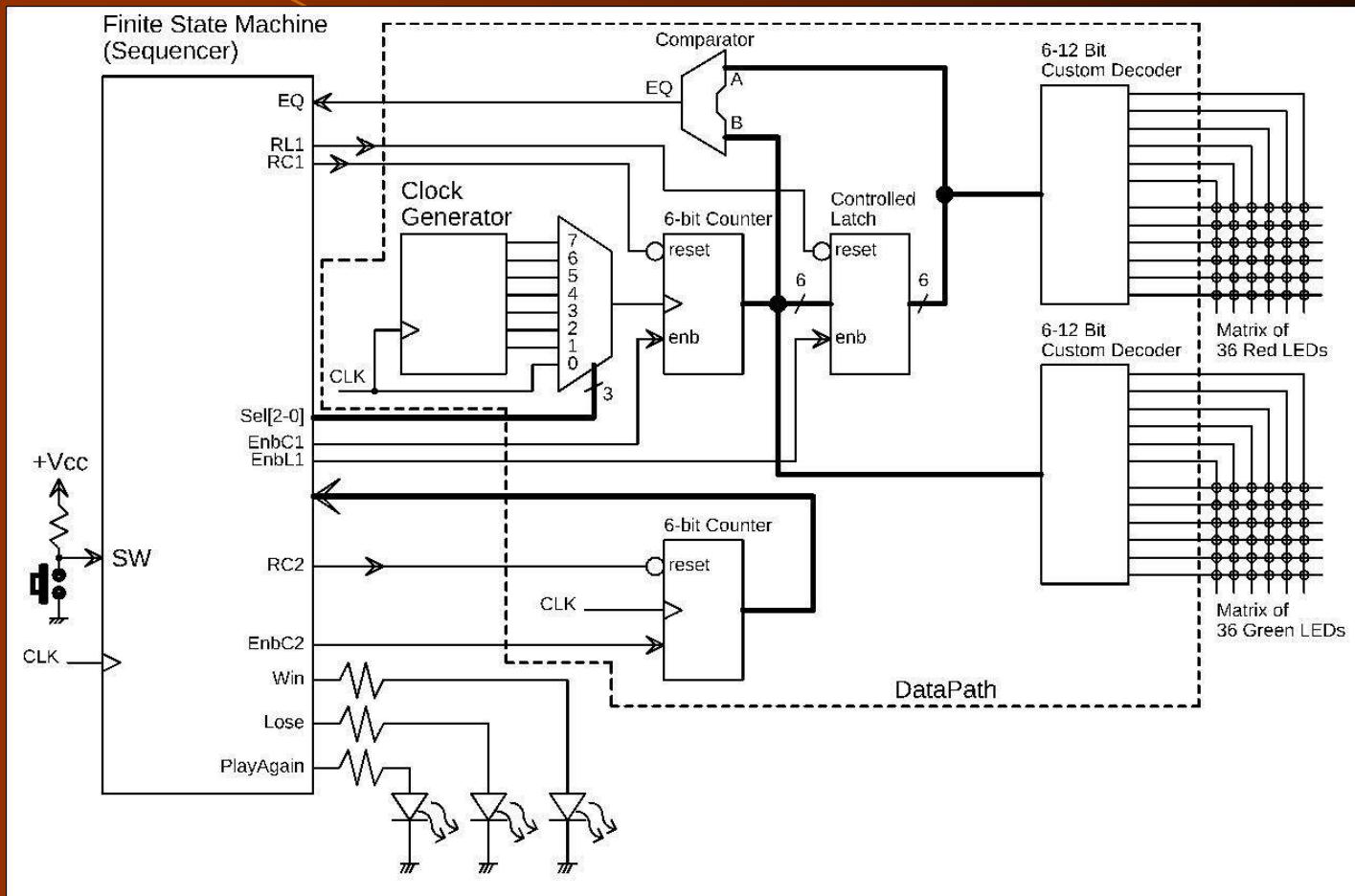


Photograph of the Arduino Based Roulette Wheel Game



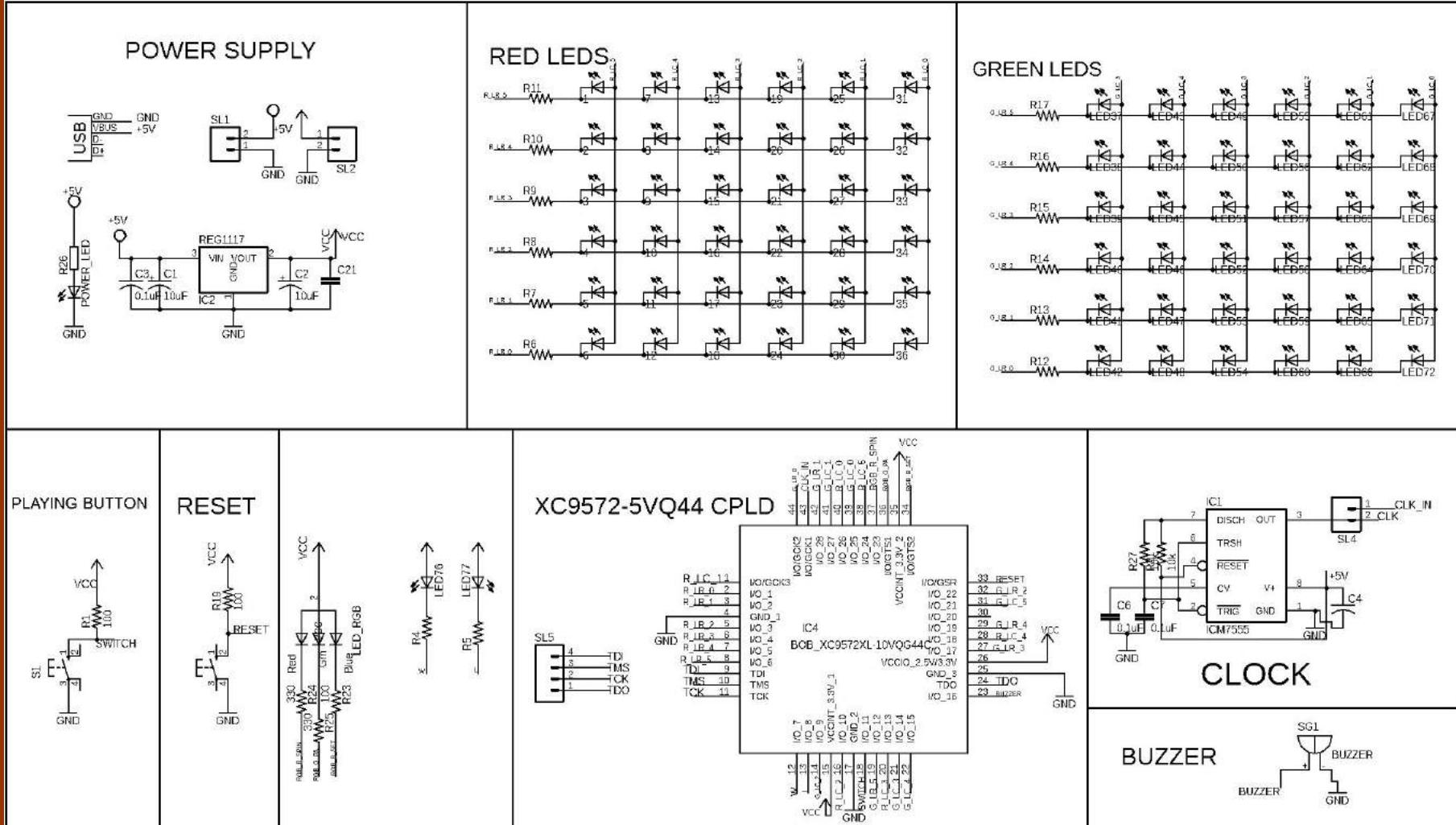
Single Purpose Computer Implementation

Block diagram of CPLD Based Roulette

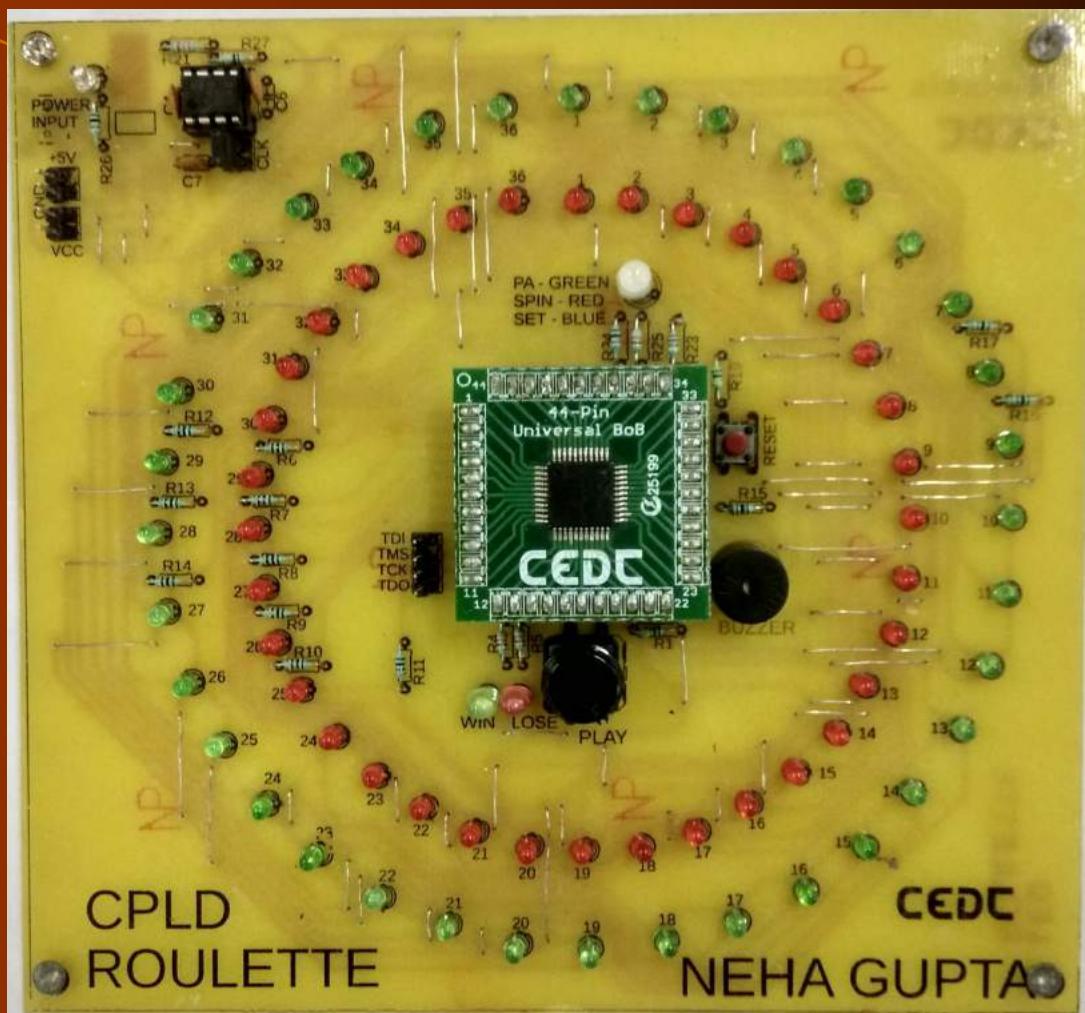


Single Purpose Computer Implementation

Schematic diagram of CPLD Based Roulette



Photograph of the Single Purpose Computer Implementation



Design Processes: Past and Present

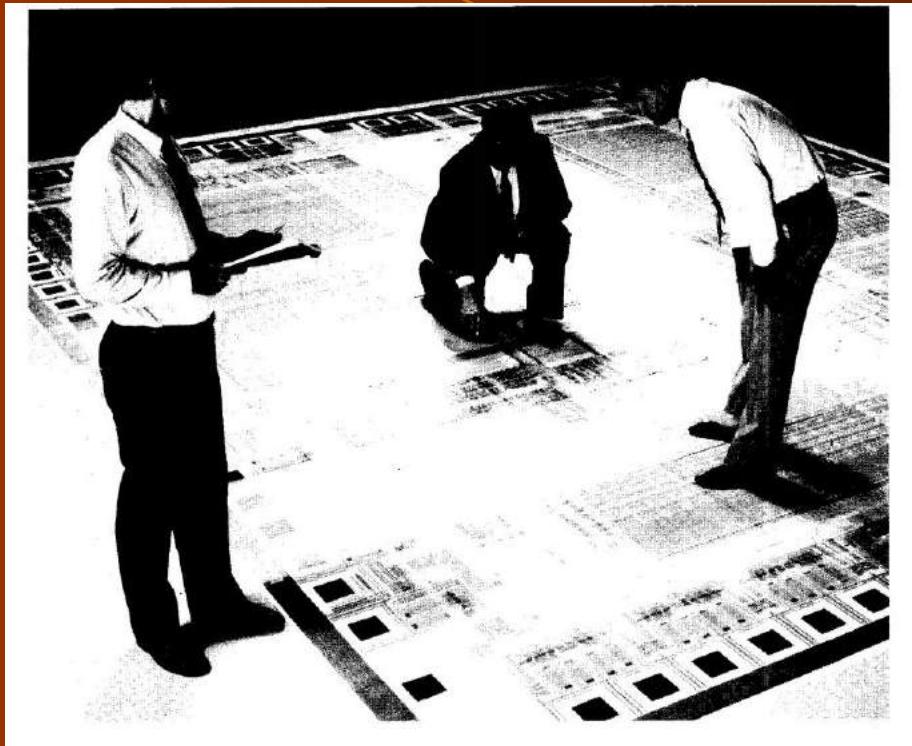


Image Source: The Art of Electronics, 2nd Edition

```
entity gl85 is
port(CLK : in bit;
      RESETOUT, SOD: out bit;
      SID, TRAP, RST75, RST65, RST55, INTR: in bit;
      INTABAR: out bit;
      A15,A14,A13,A12,A11,A10,A9,A8: out bit;
      AD7,AD6,AD5,AD4,AD3,AD2,AD1,AD0: out bit;
      S0, ALE, WRBAR, RDBAR, S1, IOMBAR: out bit;
      READY, RESETINBAR: in bit;
      CLKOUT, HLDA: out bit;
      HOLD : in bit;
      DIN : in bit_vector(7 downto 0);
      VCC,GND: in bit);
end;
|
architecture structure of gl85 is
begin
  ID<->I0,I1,I2,I3,I4,I5;
  B8BO,B8BIN<->B8BIN;
  B16BO,B16BIN<->B16BIN;
  T1,T2,T3,T4,T5,T6<->T1;
  ACCOUTEN,WRACC,WR2TEMP,WRAUXACC,ENBUSTOAUX<->INA,LASTMC,CC6,CCBAR;
  RSTN,INTA,VINT,THALT,THOLD,TWAIT<->RSTN;
  M1,M2,M3,M4,M5<->M1;
  MDRROUT,BIMC,ALUOUTEN,CC<->MDROUT;
  CLKOUT_buf,TEMP_OUT<->CLKOUT_buf;
  SEL16BUS,SEL_CNTR<->SEL16BUS;
  WRB,WRC,WRBC,WRPCH,WRPCL,WRPC<->WRB;
  BOUT,COUT,BCOUT,PCHOUT,PCLOUT,PCOUT,WRH,WRI,WRHI
```

Synthesizable VHDL Code for 8085 Microprocessor

Embedded System Implementation at System Level

- Build Your Own. Good for volume applications.
- COTS (Components Off the Shelf) Approach.

eg. Use a regular PC Motherboard and custom application program. Use Raspberry Pi or BeagleBone Black SBC. Good for small quantity.

Demonstration of MSP430 Based Projects

Lecture - 1 Summary

Introduction to Embedded Systems

- Course Objectives, Prerequisites, Logistics for the course
- Demonstration of MSP430 Based projects
- Formal definition of Embedded Systems. Everyday examples of embedded Systems - household gadgets etc.
- (a) Physical Computing (b) Cyber Physical Systems (c) IoT and (d) Embedded Systems as closely related topics
- Comparing Characteristics of Embedded Systems and General Purpose Computing Systems.
- Microprocessor, Microcontroller and SoC Terminology
- Embedded System implementation: i). Processor level using (a) generic devices (b) full custom ASIC and c) single purpose computers ii). System Level



Thank you!

Introduction to Embedded System Design

Lecture - 2: Modular Approach to Embedded System Design

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Key Parameters for a Successful Embedded System

- Time to Market (4 week delay can lead to 30% drop in revenues)
- Overall System Cost

Choosing the Right Microcontroller

- Time to Market
 - Is the Microcontroller easy to use?
 - High Level Language Supported?
- How Difficult to move to a different Device?
 - Does a compatible device with more/less memory exist?
- What about Development Tools?
 - assembler, compiler, debugger, emulators, eval kits?
- Support?

Choosing the Right Microcontroller-II

- Overall System Cost

Cost of Microcontroller + external components

Cost of PCB

- Does the Microcontroller offer higher Integration upgrade?

more software features could be accommodated

- Hidden Costs?

stocking multiple devices, turn-around time, upgrade?

Microcontroller Classification

1. Memory Architecture:

- Von Neumann
- Harvard

2. Bit Handling Capacity

- 4, 8, 16, 32, 64-bits

3. Instruction Set Architecture

- CISC: Complex Instruction Set Computing
- RISC: Reduced Instruction Set Computing
- MISC: Minimal Instruction Set Computer
- VLIW: Very Large Instruction Word

Current Favorites – The 8-bitters

- 8051 family – more than 1000 variants, in varied packagings. Standard CISC
- Microchip's PIC – RISC architecture.
- Cypress Semiconductor PSoC. CISC, complete system on chip with programmable analog blocks
(<https://www.cypress.com/>)
- Microchip's (ex-Atmel) AVR. RISC. 200+ chips

The 16-bitters

- Texas Instruments MSP430
- Microchip PIC24
- STMicroelectronics ST10
- NXP HC12 (Legacy), HC16

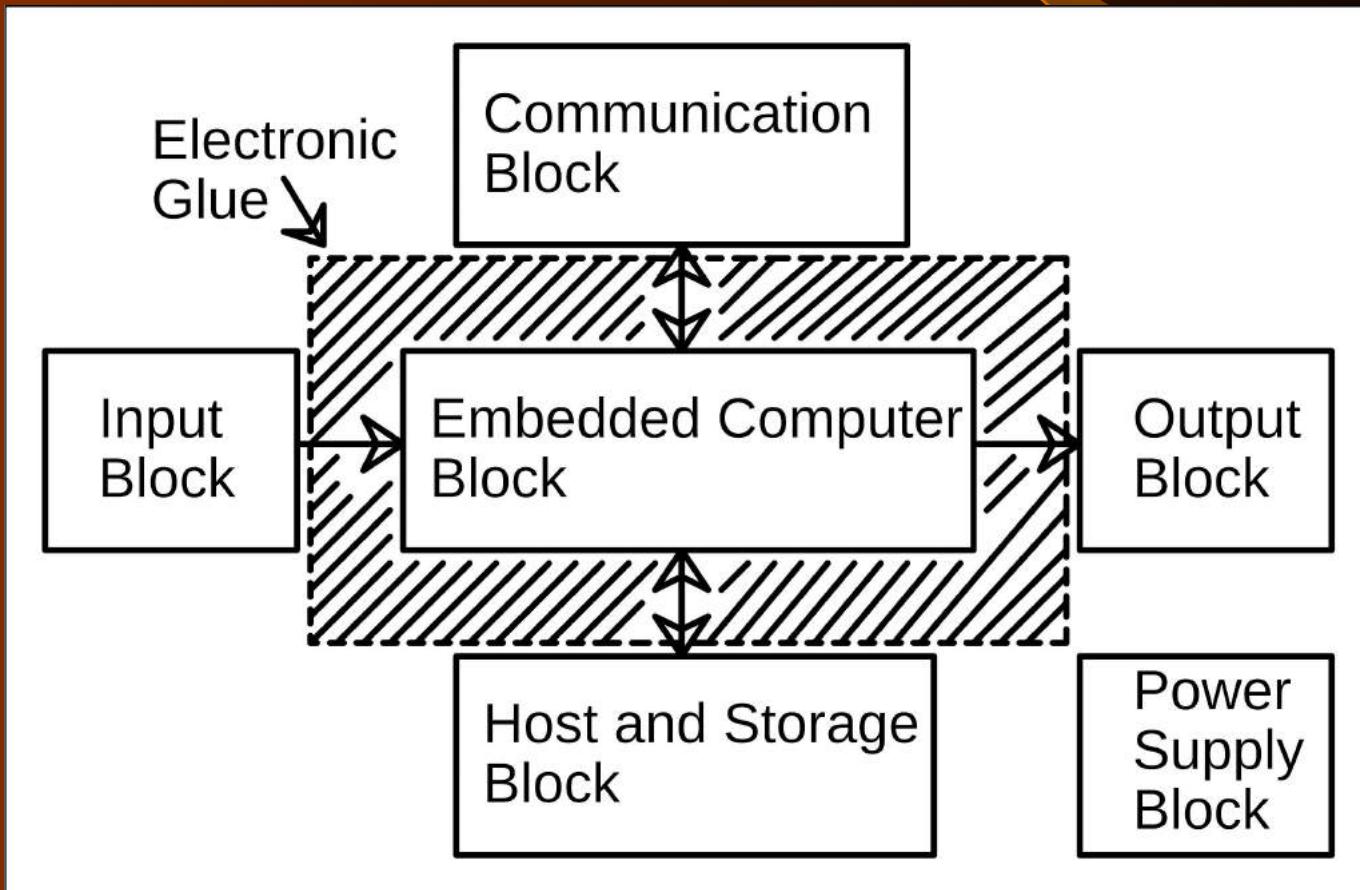
The 32-bitters

- 16/32-bit ARM family, 1000s of variants.
- Intel x86 family
- IBM's PowerPC (used in telecom apps)

Modular Approach to an Embedded System Design

Six Box Model for Embedded System Design

Any embedded device correlates to this generic model



Input Block

- **User Input**

Push Button, Toggle Switch,
SPST/SPDT/MPMT selector
switches, Switch Matrix, Capacitive
touch, Resistive touch, Reed switch
(with a magnet input)

- **Sound**

Microphone, Ultrasonic

- **Magnetic Field**

Hall Effect, Inductor, Reed switch,
Magnetometer

- **Distance**

Ultrasonic ranger, IR proximity
sensor

- **Temperature**

Thermistor, RTD, Thermocouple,
Semiconductor Sensor

- **Light**

LDR, Photodiode, LED as
sensor

- **Strain/Force**

Strain gauge, FSR, Piezo

- **Relative Position**

Shaft encoder (Stepper Motor
as a shaft encoder),
Gyroscope, Optocoupler,
Linear potentiometer, GPS

- **Image**

Camera (CMOS or CCD),
Linear CCD array

- **Time**

RTC, Clock + Counter

Output Block

- **Light**
LED, RGB LED, Laser, IR
- **Visual**
Seven Segment/Alphanumeric Display,
Character LCD, Graphics LCD, TV
- **Sound**
Speaker, Buzzer, Ultrasound
- **Temperature**
Heater, Peltier module
- **Position**
Stepper Motor (Microstepping mode),
DC Motor, Servo Motor, Servo
mechanism, Solenoid
- **Flow**
Valve, Pump
- **Haptic**
Vibration (Motor + asymmetric load)
- **Print**
Thermal printer, Dot-matrix printer

Power Supply Block

- Energy Source?
- Regulator: Linear or Switching?
- If Switching, then – Buck, Boost, Buck-Boost?
- Battery technologies?
- Supercapacitor?

Communication Links Block

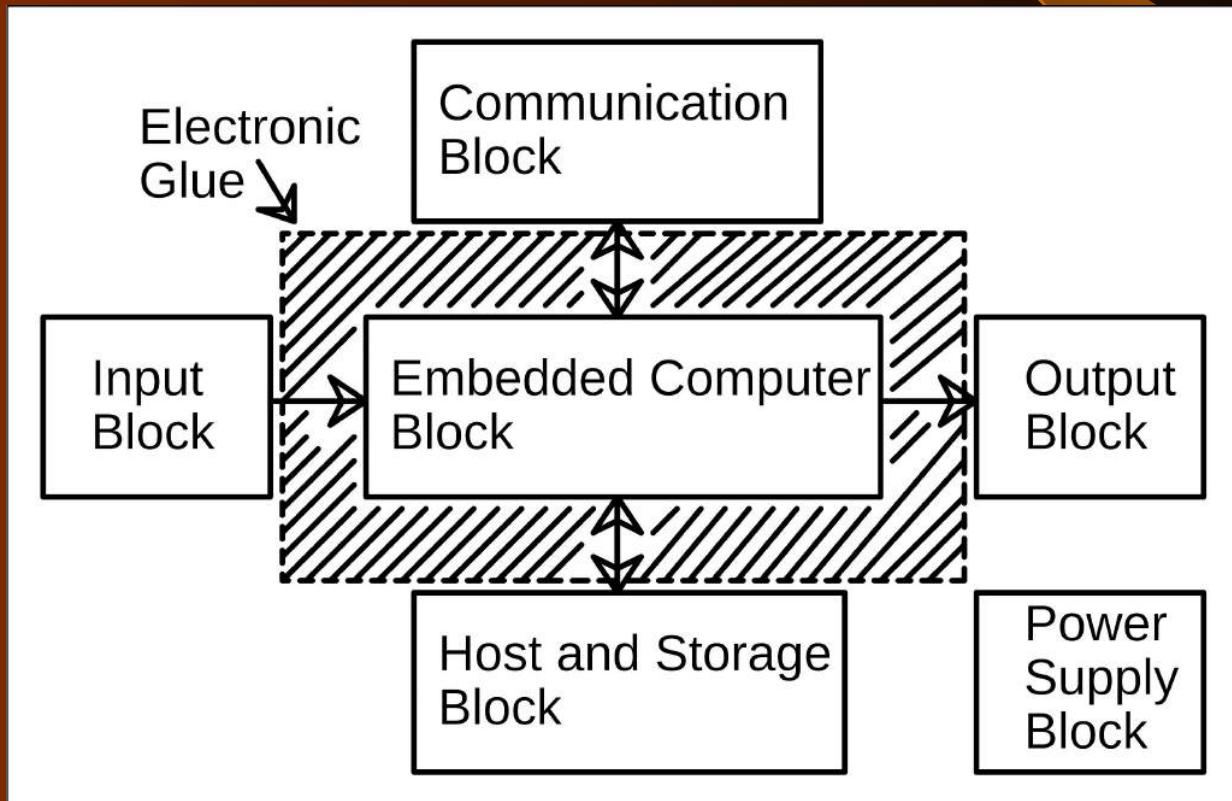
- Inter-device Vs Intra-device
- Intra-device: **UART, SPI, I2C**
- Inter-device: **UART, LIN (Local Interconnect Network), CAN, WiFi, Ethernet, USB, Bluetooth**

Host and Storage Block

- Serial E2PROM
- SD Card

Electronic Glue

- Analog front end: Amplifiers, filters.
- Output: Power Switching (Low, high and both side switching)



Lecture - 2 Summary

- Key parameters of Embedded System Design
- Time to market and cost.
- Microcontroller Classification based on memory access, ISA, data bus width (Example Microcontroller families (8-bit, 16-bit and 32-bit examples)), Memory technologies, Memory interface busses.
- Modular approach to Embedded System Design using a Six Box Model of an Embedded System: Input, Output, Processor, Power Supply, Communication, Host.



Thank you!

Introduction to Embedded System Design

Lecture - 3: Microcontroller Features, Essential Elements of Microcontroller Ecosystem

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Salient Features of Modern Microcontrollers

Expected Functions from a Microcontroller

- Read Digital Inputs
- Provide Digital Outputs
- Measure or maintain a record of Time (relative or absolute)
- Measure time between two events
- Measure Duration of an event
- Generate random numbers

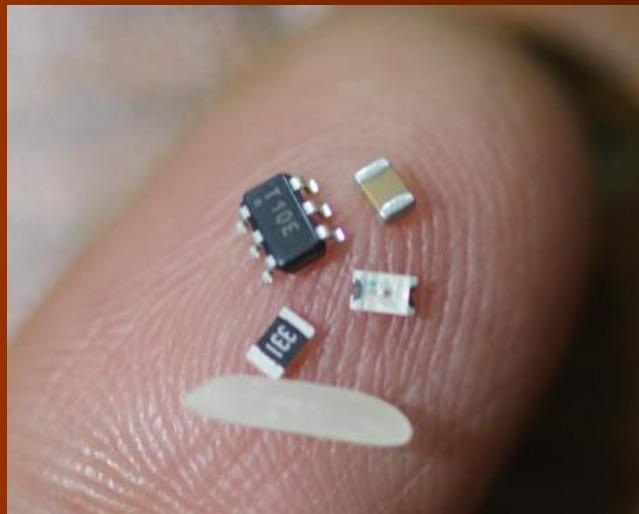
Expected Functions from a Microcontroller

- Respond to asynchronous events (Interrupts)
- Measure Voltage/Current/Resistance
- Provide analog voltage/current
- Store data
- Visualize data/information
- Print data
- Control motion

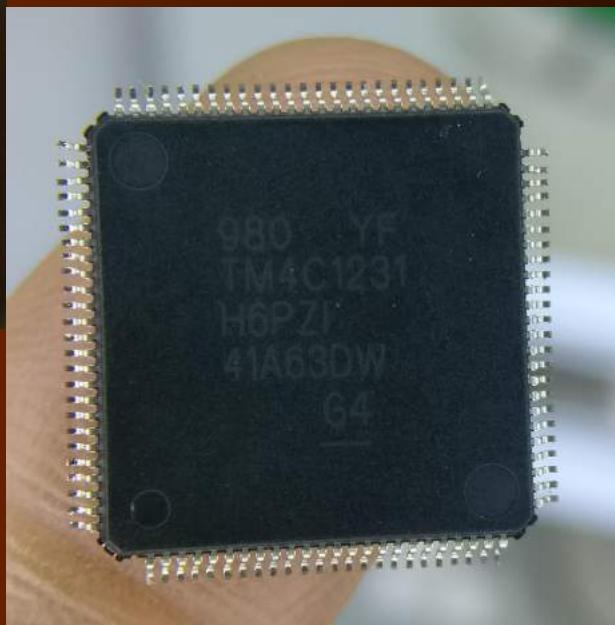
Microcontrollers

- CPU – 4/8 /16 /32/64 bit microprocessor (CISC / RISC, VN/Harvard)
- Memory
 - ROM / EEPROM / Flash memory – Program storage
 - Volatile Memory (RAM) – Data Storage
- Digital Input / Output pins
- Communication Interfaces
- In System Programming & Debugging
 - SPI Bus/JTAG Interface
- Peripherals
 - Timers, Counters, PWM Generators
 - Watchdog Timer with independent oscillator
 - Analog to Digital Converter, Digital to Analog Converter (explicit or implicit)
- 6 Pin to 200+ pin Devices!

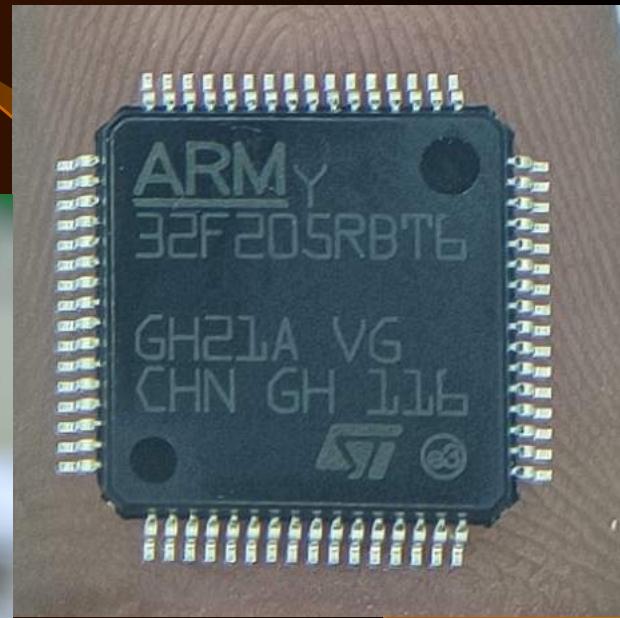
Size Diversity



AT Tiny10



TIVA TM4C1231
Cortex-M4



STM32F205
Cortex-M3

Features of Modern Microcontrollers

- Fully programmable pins – Input or Output
- Output pins with 30-40 mA Source/Sink current capability
- Input Pins with weak and strong pull up or tri-state capability
- Each pin offers multiple functions one of which can be selected

Multiple Functions on Each Pin

Device Pinout, MSP430G2x13 and MSP430G2x53, 28-Pin Devices, TSSOP

DVCC	1	DVSS	28
P1.0/TA0CLK/ACLK/A0/CA0	2	XIN/P2.6/TA0.1	27
P1.1/TA0.0/UCA0RXD/UCA0SOMI/A1/CA1	3	XOUT/P2.7	26
P1.2/TA0.1/UCA0TXD/UCA0SIMO/A2/CA2	4	TEST/SBWTCK	25
P1.3/ADC10CLK/CAOUT/VREF-/VEREF-/A3/CA3	5	RST/NMI/SBWTdio	24
P1.4/SMCLK/UCB0STE/UCA0CLK/VREF+/VEREF+/A4/CA4/TCK	6	P1.7/CAOUT/UCB0SIMO/UCB0SDA/A7/CA7/TDO/TDI	23
P1.5/TA0.0/UCB0CLK/UCA0STE/A5/CA5/TMS	7	P1.6/TA0.1/UCB0SOMI/UCB0SCL/A6/CA6/TDI/TCLK	22
P3.1/TA1.0	8	P3.7/TA1CLK/CAOUT	21
P3.0/TA0.2	9	P3.6/TA0.2	20
P2.0/TA1.0	10	P3.5/TA0.1	19
P2.1/TA1.1	11	P2.5/TA1.2	18
P2.2/TA1.1	12	P2.4/TA1.2	17
P3.2/TA1.1	13	P2.3/TA1.0	16
P3.3/TA1.2	14	P3.4/TA0.0	15

NOTE: ADC10 is available on MSP430G2x53 devices only.

Multiple Functions on Each Pin

(PCINT14/RESET) PC6	<input type="checkbox"/>	1	28	<input type="checkbox"/> PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	<input type="checkbox"/>	2	27	<input type="checkbox"/> PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	<input type="checkbox"/>	3	26	<input type="checkbox"/> PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	<input type="checkbox"/>	4	25	<input type="checkbox"/> PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	<input type="checkbox"/>	5	24	<input type="checkbox"/> PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	<input type="checkbox"/>	6	23	<input type="checkbox"/> PC0 (ADC0/PCINT8)
VCC	<input type="checkbox"/>	7	22	<input type="checkbox"/> GND
GND	<input type="checkbox"/>	8	21	<input type="checkbox"/> AREF
(PCINT6/XTAL1/TOSC1) PB6	<input type="checkbox"/>	9	20	<input type="checkbox"/> AVCC
(PCINT7/XTAL2/TOSC2) PB7	<input type="checkbox"/>	10	19	<input type="checkbox"/> PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	<input type="checkbox"/>	11	18	<input type="checkbox"/> PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	<input type="checkbox"/>	12	17	<input type="checkbox"/> PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	<input type="checkbox"/>	13	16	<input type="checkbox"/> PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	<input type="checkbox"/>	14	15	<input type="checkbox"/> PB1 (OC1A/PCINT1)

Features of Modern Microcontrollers

- Some functions available on multiple pins.
- Flash memory for Program storage (1KB to few MB) with lock
- SRAM for data (0 to 10s of KB)
- EEPROM for semi-constants (0 to few KB)

Functions Available on Multiple Pins

Table 16-1. I²C Signals (100LQFP)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type ^a	Description
I ² C0SCL	72	PB2 (3)	I/O	OD	I ² C module 0 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I ² C0SDA	73	PB3 (3)	I/O	OD	I ² C module 0 data.
I ² C1SCL —	34 74	PA6 (3) PG4 (3)	I/O	OD	I ² C module 1 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I ² C1SDA —	35 75	PA7 (3) PG5 (3)	I/O	OD	I ² C module 1 data.
I ² C2SCL —	36 95	PF6 (3) PE4 (3)	I/O	OD	I ² C module 2 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I ² C2SDA —	58 96	PF7 (3) PE5 (3)	I/O	OD	I ² C module 2 data.
I ² C3SCL	1 62	PD0 (3) PG0 (3)	I/O	OD	I ² C module 3 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.

Features of Modern Microcontrollers

- Interrupt on Pin Change on all/most pins
- Internal and external clock sources. Internal RC clock w/ trimming
- Clock scalability
- Multiple Vcc domains
- Low operating current (from 1mA/MHz to 0.1 mA/MHz)

Features of Modern Microcontrollers

- Operating modes: active, sleep, power down
- Wide Vcc range (0.9V to 6V)
- Multiple Reset sources: POR, User, BOD, Watchdog
- Watchdog timer with independent clock
- BOD with programmable threshold

Features of Modern Microcontrollers

- Mixed signal capability – SAR/Dual-slope/Delta-Sigma ADC with resolution of 8 to 16 bits
- Mixed signal capability – hardware DAC or through PWM
- Mixed signal requirement – Independent Analog & Digital Vcc
- Timer and Counter: 8, 16-bits.
- High frequency PWM with various options (dead zone etc.)

Features of Modern Microcontrollers

- General Purpose Communication: UART, SPI, I2C, 1-wire
- Specialized Communication: CAN, LIN, Ethernet, USB (device, OTG, Host), WiFi, Bluetooth, BLE, RF
- Special function: floating point unit, encryption unit, hardware accelerator
- In System Programming and In Application Programming
- On Chip Debug
- JTAG
- DMA capability on needed peripherals

Application Development Tools

- Development platform (usually a PC these days)
- Assembler (cross-assembler), Compiler (cross-compiler)
- Simulator/debugger
- Evaluation board
- Program Download tool (ISP, or Bootloader)
- Emulator and/or On Chip Debugger (OCD)

System Development Tools

- Circuit prototyping facility
- Schematic capture and PCB layout CAD tools
- Circuit Soldering (POC, final versions) facility
- Microscope/Magnifier
- Oscilloscope, Logic Analyzer, Mixed Signal Oscilloscope
- DMM, LCR bridge
- Bench power supply

Elements of Microcontroller Ecosystem

“Roti, Kapda, Makaan and Internet” for a Microcontroller! (Essential Elements for Survival)

- Clock
- Reset
- Power Supply
- Program Download Capability

The Clock Subsystem!

- Why do we need Clock?
- What Should be the Clock Frequency?
- Implications of Clock Frequency Value?
- What Topology for the Clock Generator?
- Desirable Features for the Clock Generator?
- RTC Clock?
- Clock Frequency Stabilization: TCXO, Temperature Sensor + Varactor diode in parallel to Crystal.

Lecture - 3 Summary

- Expected Functions From a Microcontroller
- Salient features of modern Microcontrollers
- Elements of Microcontroller ecosystem



Thank you!

Introduction to Embedded System Design

Lecture - 4: Elements of Microcontroller Ecosystem, Power Supply for Embedded Systems

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Elements of Microcontroller Ecosystem

“Roti, Kapda, Makaan and Internet” for a Microcontroller! (Essential Elements for Survival)

- Clock
- Reset
- Power Supply
- Program Download Capability

The Clock Subsystem!

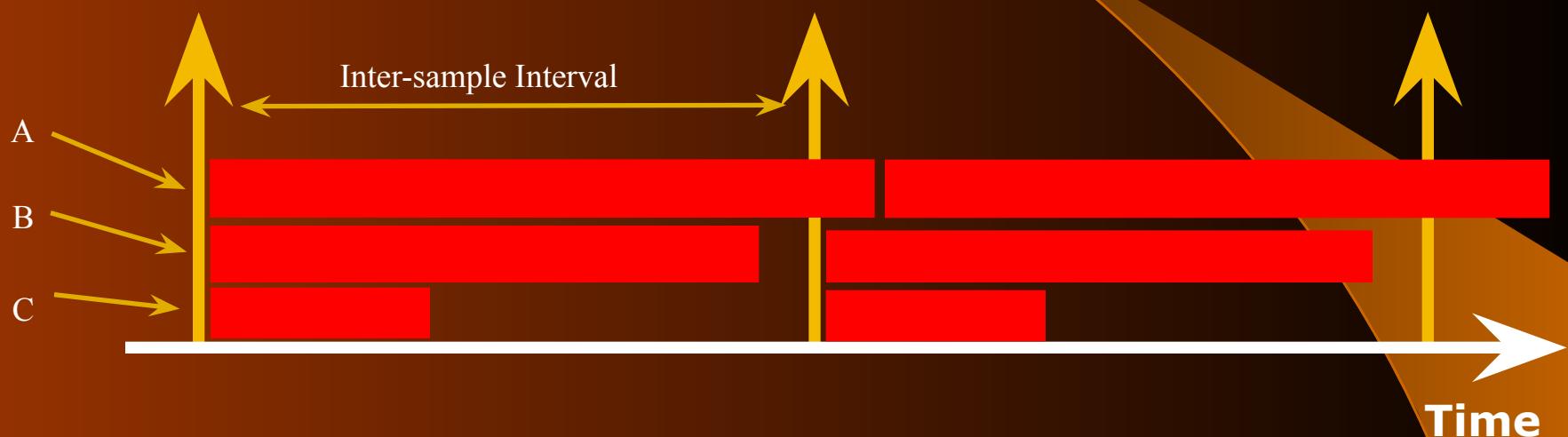
- Why do we need Clock?
- What Should be the Clock Frequency?
- Implications of Clock Frequency Value?
- What Topology for the Clock Generator?
- Desirable Features for the Clock Generator?
- RTC Clock?
- Clock Frequency Stabilization: TCXO, Temperature Sensor + Varactor diode in parallel to Crystal.

Selecting a Suitable Embedded Controller

- Peripheral Features
- Memory
- Packaging
- Grade (Commercial, Industrial, Automotive, Military)
- Price
- Availability and lead time

But most important!

Selecting a Suitable Embedded Controller



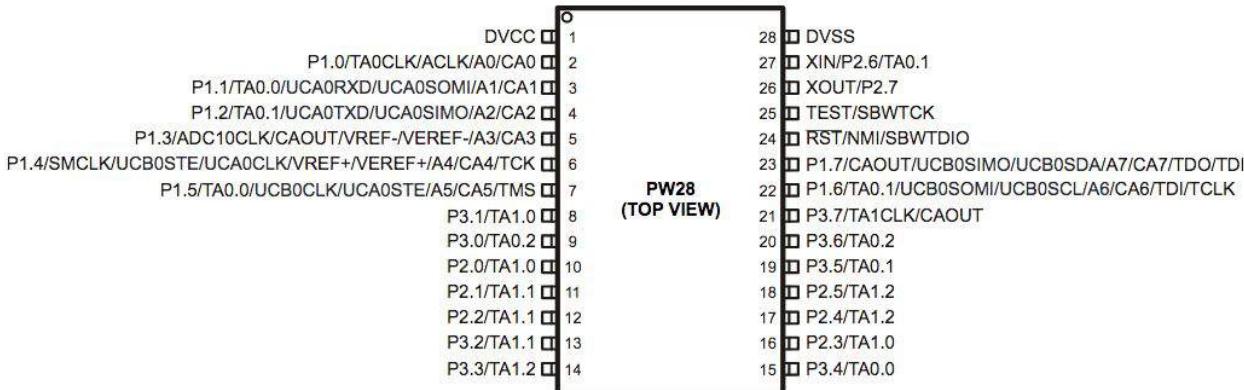
- Algorithm Complexity viz-a-viz Sampling Theorem Constraints
- Controller 'A' is unacceptable
- Controller 'B' is marginal
- Controller 'C' is comfortable!

The Reset Subsystem!

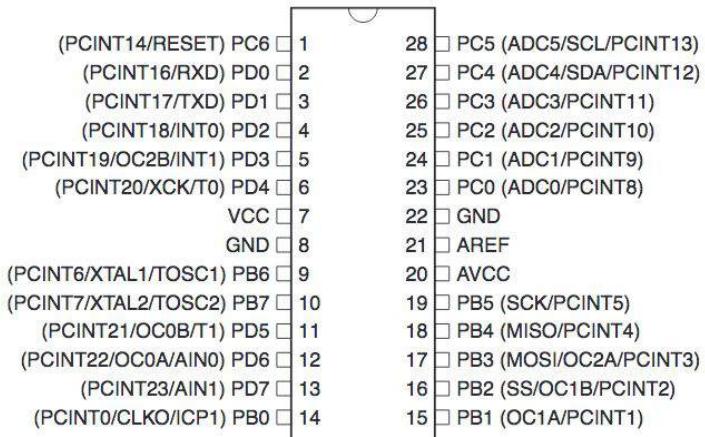
- Why do we need Reset?
- Sources of Reset?
- Warm and Cold Reset?
- POR, User, BOD and Watchdog.

User Reset

Device Pinout, MSP430G2x13 and MSP430G2x53, 28-Pin Devices, TSSOP



NOTE: ADC10 is available on MSP430G2x53 devices only.

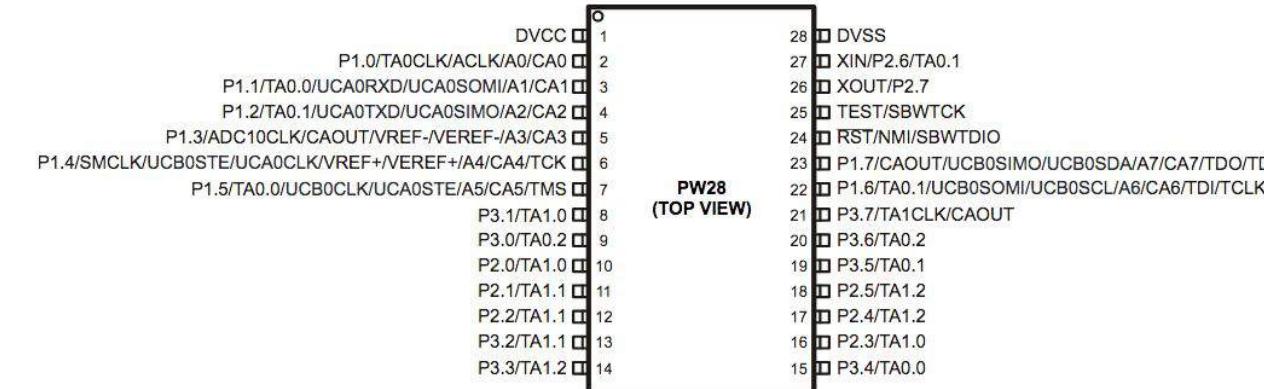


Program Download Capability

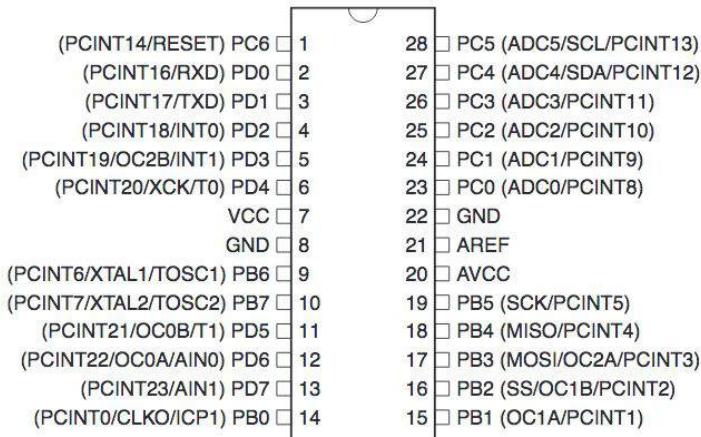
- Older Methods (EPROM Eraser + Universal Device Programmer).
- Evolution of Flash Memory → ISP.
- Most Popular using SPI.
- JTAG: Dual use. Testing as well as Program Download
- SWD
- IAP using Bootloader.
- UPM + BPM.

Power Supply for microcontrollers

Device Pinout, MSP430G2x13 and MSP430G2x53, 28-Pin Devices, TSSOP



NOTE: ADC10 is available on MSP430G2x53 devices only.



Power Supply: An Important Component

- Source of Power? - Grid, Battery or alternative?
- Stabilization (Voltage Regulation)
- Backup?
- Optimization- modes of operation

Power Supply: An Important Component

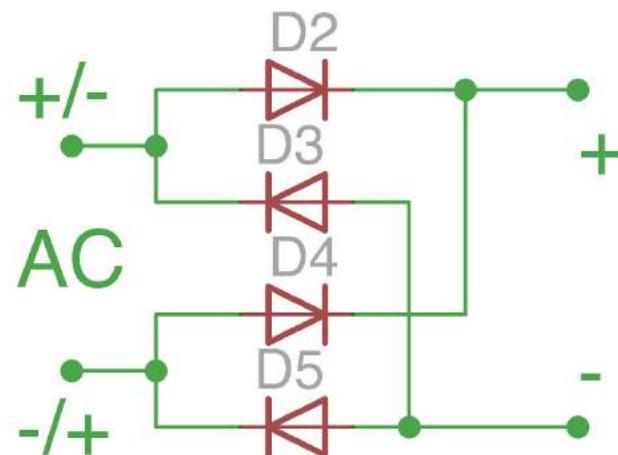
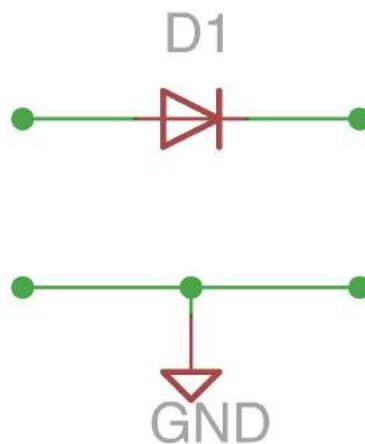
Alternative sources:

- Solar (Photovoltaic, Thermoelectric)
- Hydro
- Vibration/Resonance based on Faraday law
- Ambient RF power conversion.

Grid Power

- AC Step down → Rectifier.
- Polarity proof input stage.
- Instead of step-down transformer,
How about capacitive voltage
divider option? Advantages?
- Switching Mode Supply?
- Filtering

Low Voltage AC Input: Polarity Safe or Polarity Proof Input Stage



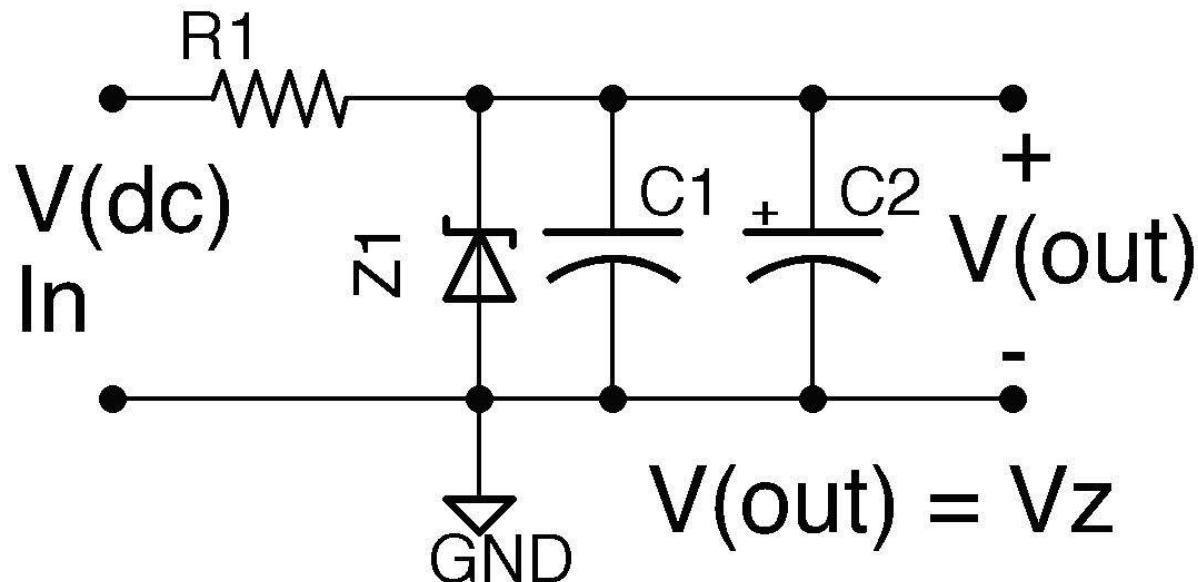
Voltage Regulators

- Linear Regulator
- Switching Regulator

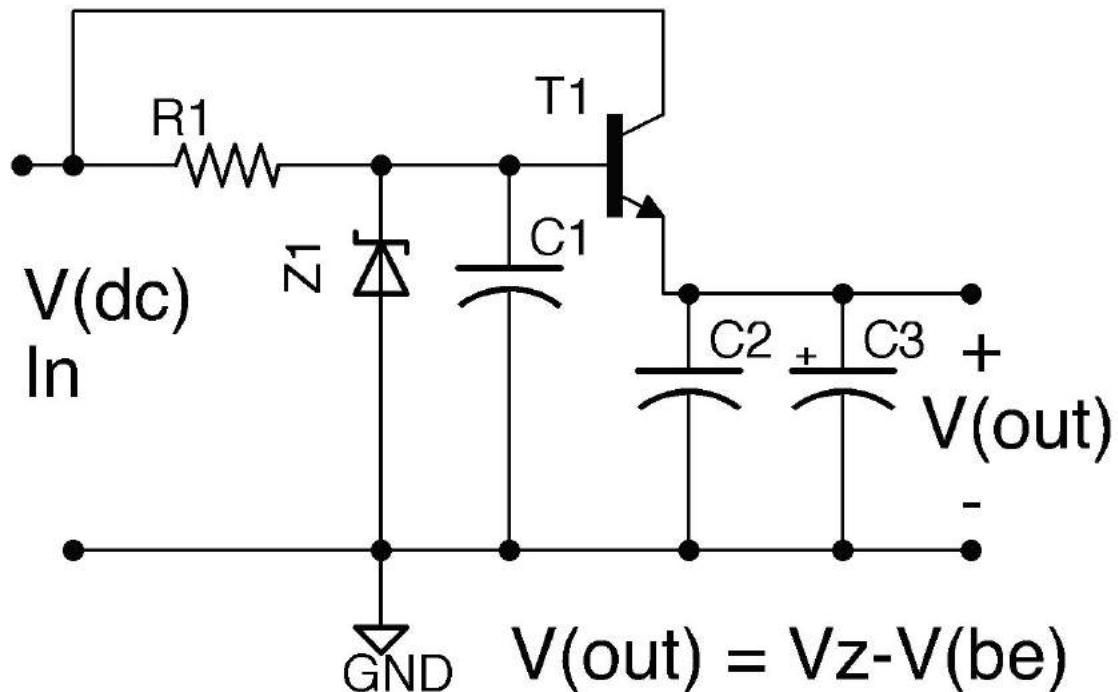
Voltage Stabilization

- Linear Regulators- Ease of use, cheap, low component count. The classic 78xxx series.
Issues? Large Dropout voltage, Large quiescent current $I(q)$.
- Use Low Drop Out (LDO) linear regulators.
- Low Quiescent Current too.

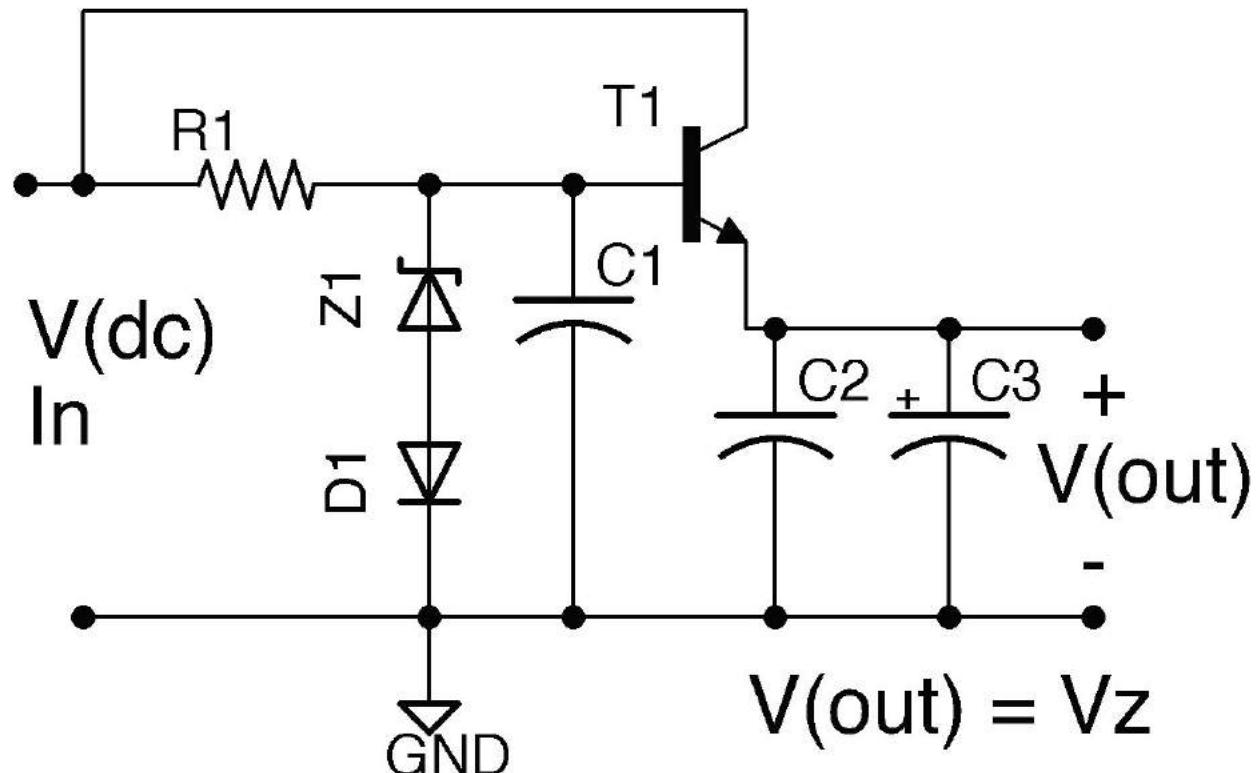
Linear Regulator Topologies



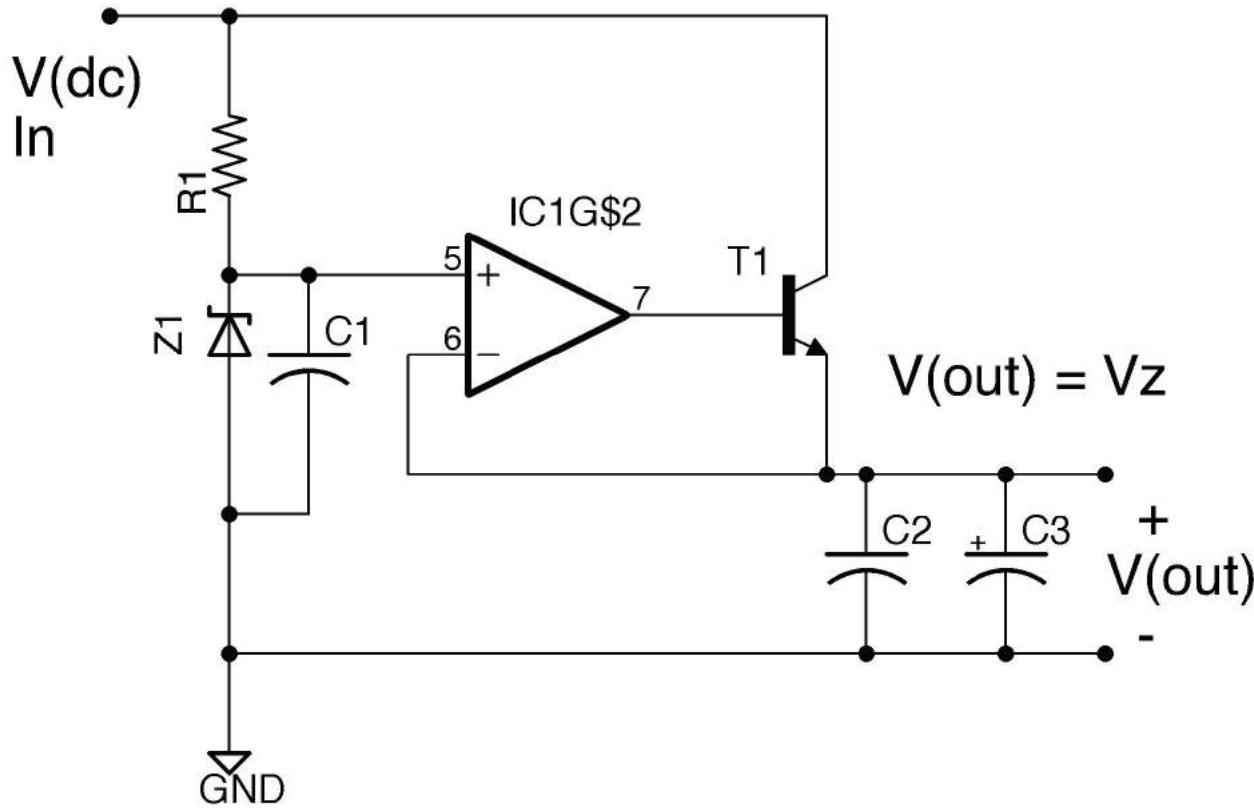
Linear Regulator Topologies



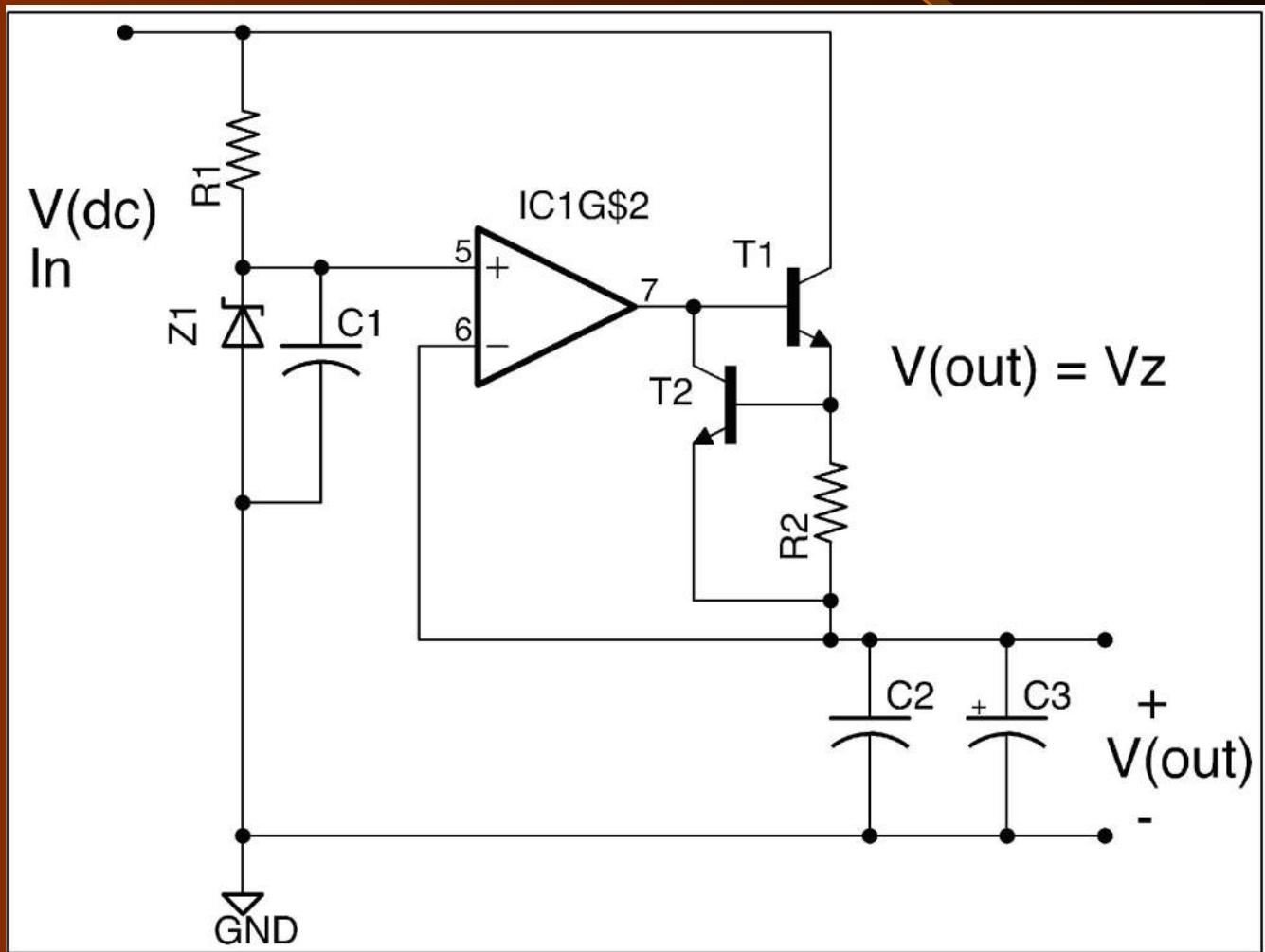
Linear Regulator Topologies



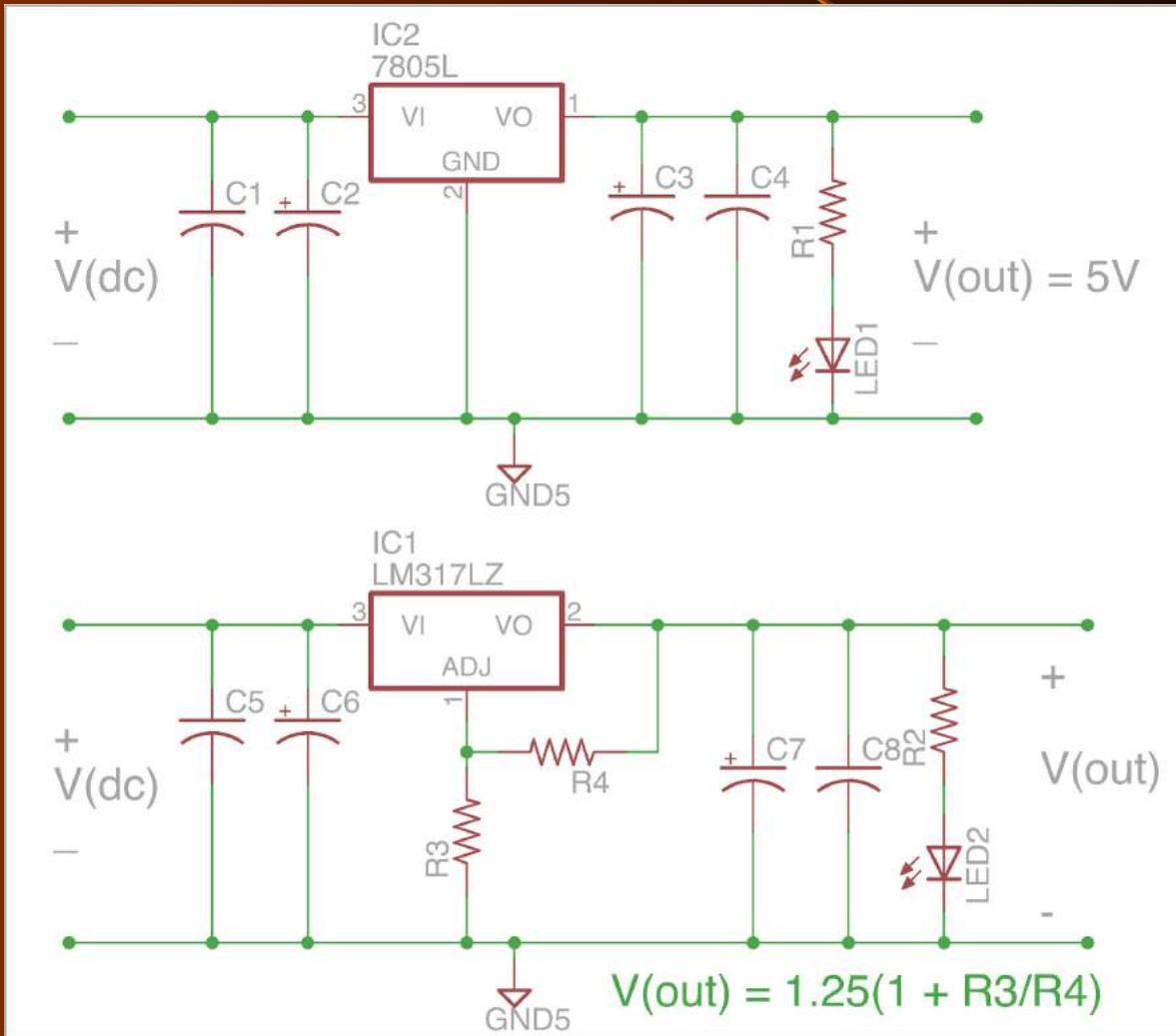
Linear Regulator Topologies



Linear Regulator Topologies



3-Terminal Linear Regulator



Linear Regulators

- The classic 7805 is history.
- Why? 5V no longer voltage of choice.
- Large Drop out voltage (3V or more)
- Large Quiescent Current - 10 mA.
- Alternatives?

LDO! 0.1V drop out, 10-100uA Quiescent Current

Examples: LP2950 (2.7, 3, 3.3V etc)

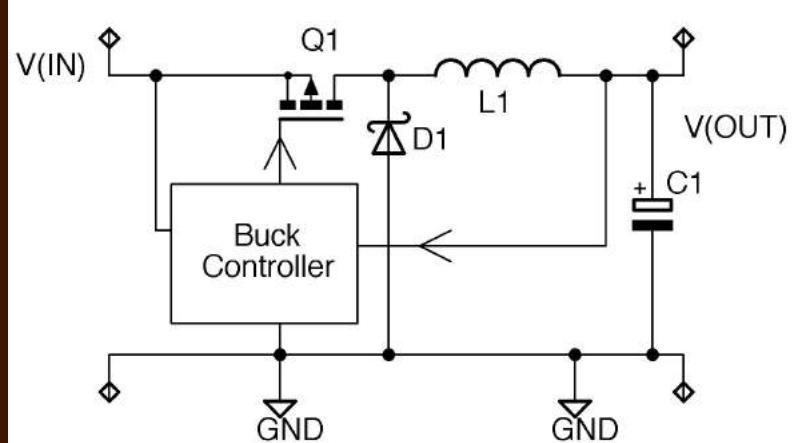
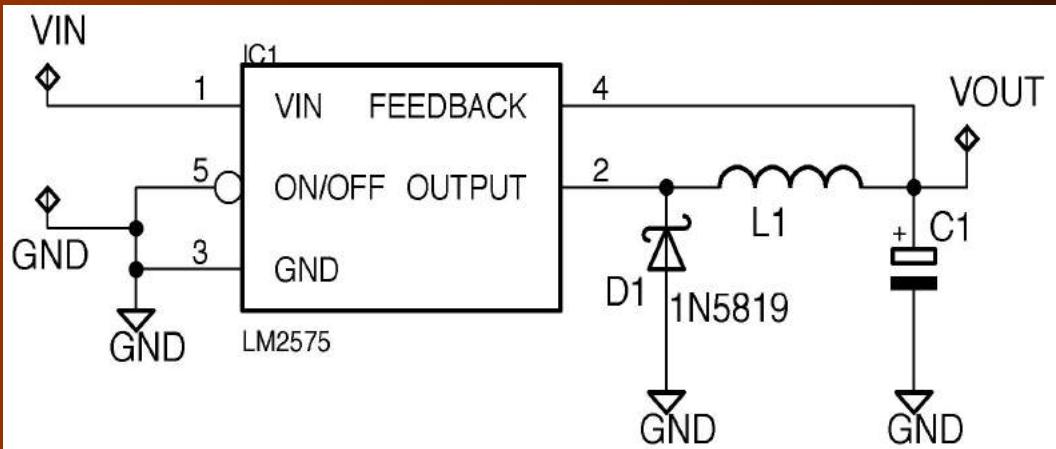
LP38500-ADJ 0.6V dropout.

Switching Regulators

- Advantages: 90% + efficiency
- Pitfalls: Inductor, switching diode, high current switch.
- Lower ESR output capacitor
- Relatively large output noise.
- Type depends on input voltage availability and output voltage requirements.

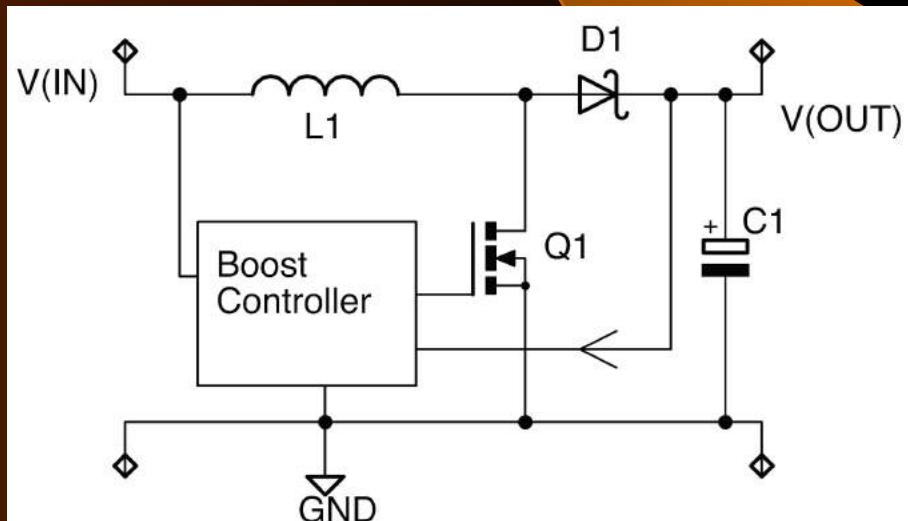
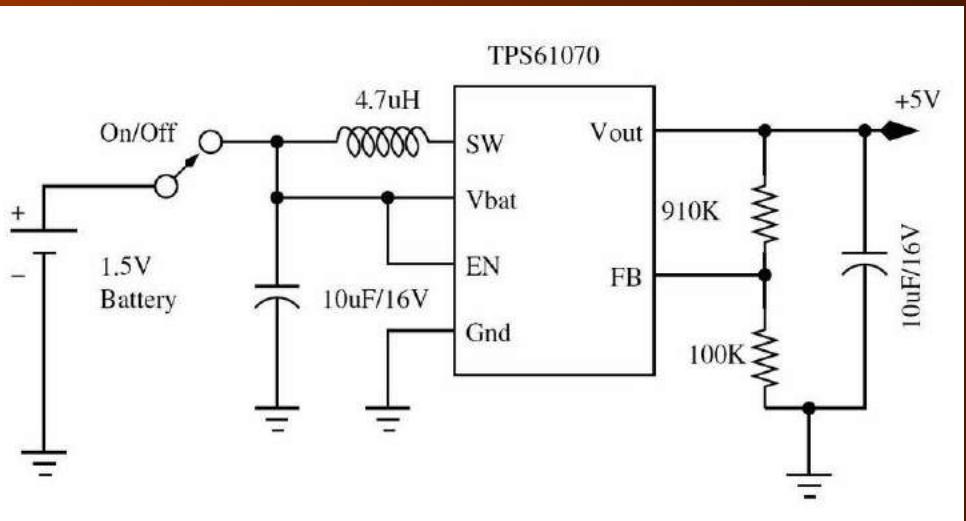
Switching Regulators - II

- Broadly 3 categories: Buck, Boost, Buck-boost.
- Buck: $V(\text{out}) < V(\text{in})$; $I(\text{out}) > I(\text{in})$
- Buck example: LM2575 (1A max); LM2576 (3A max.)



Switching Regulators - III

- Boost: $V(\text{out}) \geq V(\text{in})$; $I(\text{out}) < I(\text{in})$
- TPS61070; $V(\text{in})$ 0.9V to 5.5V; $V(\text{out}) < 5.5\text{V}$ (adjustable); Switch Current: 600mA
- LT1308: $V(\text{in})$: 1V to 10V; $V_{\text{out}} < 34\text{V}$. Switch Current: 3A.



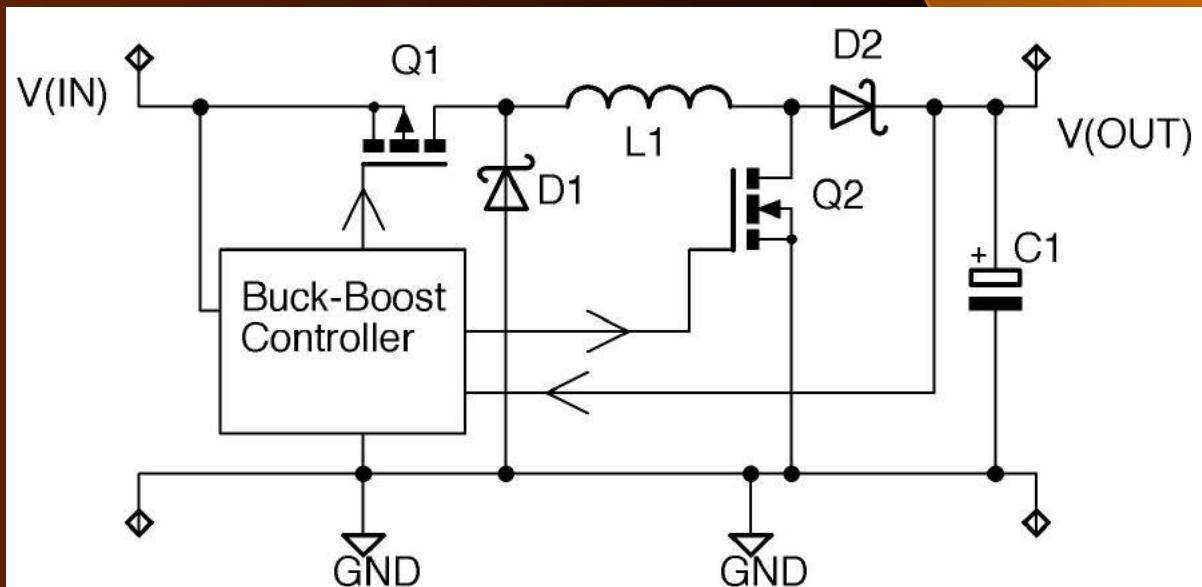
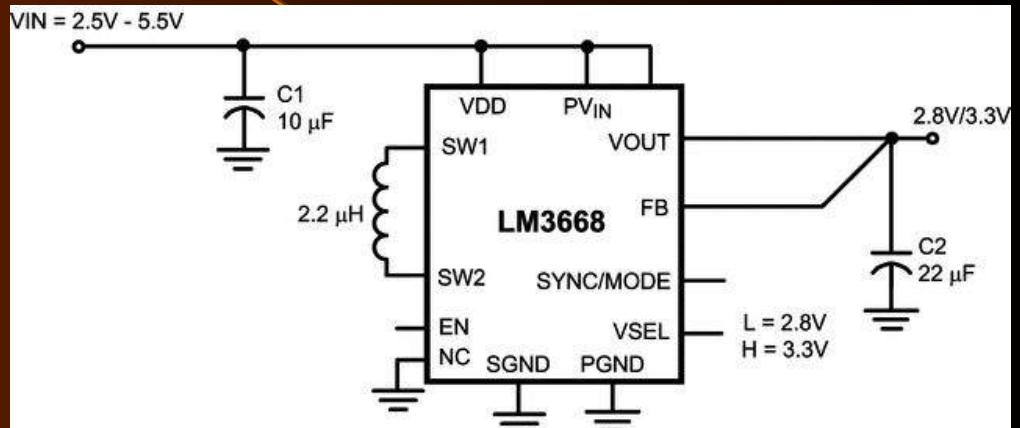
Switching Regulators - IV

- Buck-Boost:

$V_{in(min)} < V_{(out)} < V_{in(max)}$

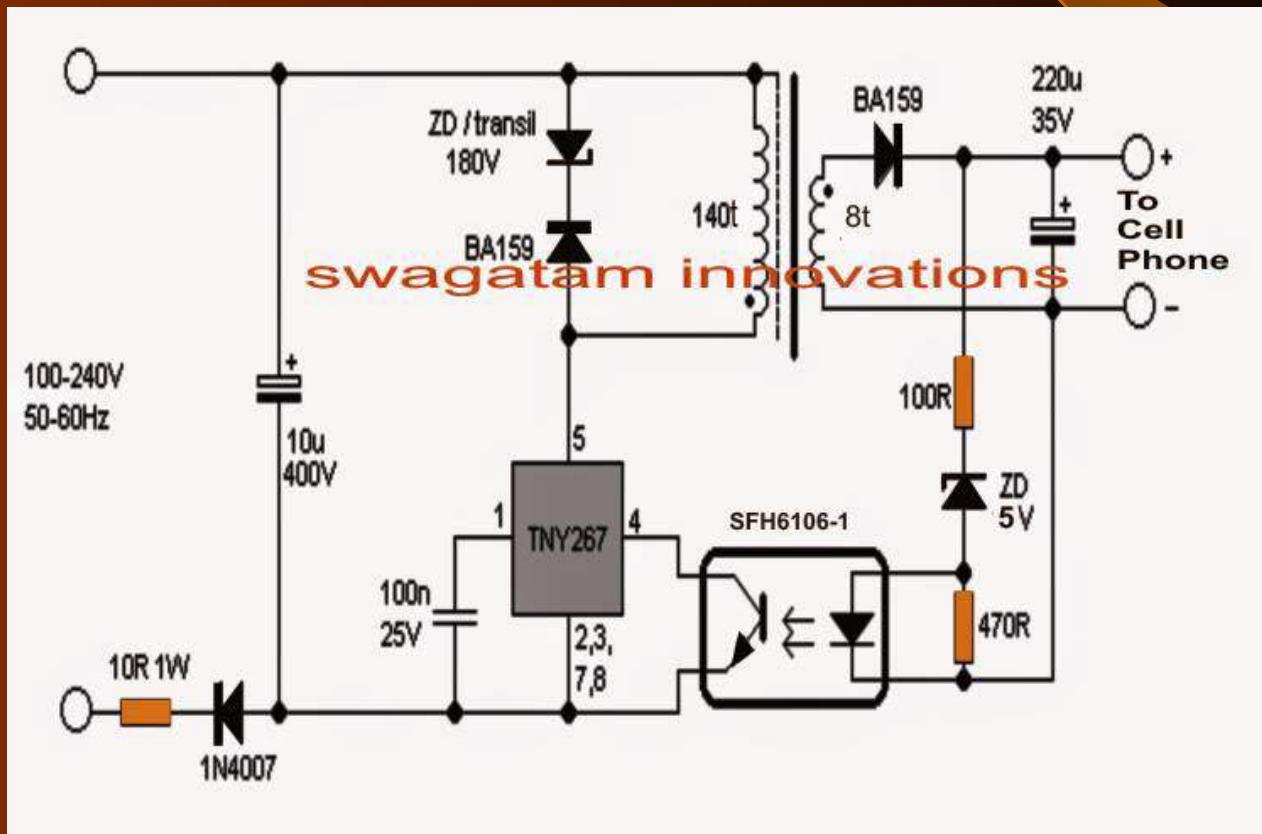
- Buck-Boost
example: LM3668.

$V_{(in)}$ 2.5V to 5.5V.
 V_{out} : 2.8V or 3.3V
(fixed). $I_{(max)}$: 1A.



Switching Cell Phone Charger

- $E(\text{avg}) = 4*f*N*a*B$



Various Power Supply Topologies

- Line Frequency Transformer + Linear (fixed or variable) regulator
- Line Frequency Transformer + switching (buck or boost) regulator
- Switching Power Supply
- Capacitive attenuator + Zener
- Alkaline + Boost/Buck

Various Power Supply Topologies - II

- Lithium + Boost (for 5V)
- Lithium + Buck-Boost (3.3V)
- Lithium + Boost for 5V and Buck/LDO for 3.3V
- Radio Frequency based Ultra low power
- Faraday based
- Solar + Battery + Regulator
- Vibration resonance (<http://www.perpetuum.com/>)
- Thermoelectric (TEG) based

Modern ICs

- Partitioned Design
- Speed Versus Noise Margin trade off
- Analog Subsystem
- All the above leads to multiple power supply rails.

Linear Power Supply for Embedded Processors

Rail	Rail Volts (V)	Est I _{OUT} (A)	P _{OUT} (W)	Cnvrtr VIN (V)	Converter Type	Linear Eff = V _o /V _{in} (%)	PIN= P _{OUT} /Eff (W)	PIN/VIN= I _{IN} Req'd (A)	Cnvrtr Power Dsptd (W)
Vcc1	1.1	0.6	0.66	3.7	Linear	30	2.22	0.60	1.56
Vcc2/I _{o2}	1.8	0.3	0.54	3.7	Linear	49	1.11	0.30	0.57
I _{o2}	3.3	0.2	0.66	3.7	Linear	89	0.74	0.20	0.08
TOTAL			0.66				4.07	1.10	

Linear + Switched Power Supply for Embedded Processors

Rail	Rail Volts (V)	Est I _{OUT} (A)	P _{OUT} (W)	Cnvrtr VIN (V)	Converter Type	Est. Eff (%)	PIN= P _{OUT} /Eff (W)	PIN/VIN= I _{IN} Req'd (A)	Cnvrtr Power Dsptd (W)
Vcc1	1.1	0.6	0.66	3.7	Switch	92	0.72	0.19	0.06
Vcc2/I _{o2}	1.8	0.3	0.54	3.7	Switch	93	0.58	0.16	0.04
I _{o2}	3.3	0.2	0.66	3.7	Linear	89	0.74	0.20	0.08
TOTAL			0.66				2.04	0.55	

Power Optimization

Broadly 3 Levels of power consumption

- Active mode – full power and performance
- Sleep mode – Oscillator On, synchronous components off. Asynchronous On. Interrupts, RTC etc On. System Restart fairly quickly. Power consumption about half of active mode.
- Power Down mode – 1/1000 of active power. All components power down; Only asynchronous components – interrupts On. Restart is slow and requires oscillator startup ~ several 100s of ms.

Power Backup Sources

Sporadic availability of power. Need to smoothen out the availability.

- Batteries - Lead-acid, NiCd, NiMH, Li, Li-poly
- Supercapacitors?

Battery Characteristics

Type	Voltage (V)	Power(W/Kg)	Eff.	Cycles	Life (yr)
Lead-Acid	2.1	180	70-90	500-800	3-20
NiCd	1.2	150	70-90	1500	-
NiMH	1.2	250-1000	66%	1000	-
Li ion	3.6	1800	99.9	1200	2-3
Li-poly	3.6	3000+	99.8	500-1000	2-3

Lecture - 4 Summary



Thank you!

Introduction to Embedded System Design

Introduction to MSP430, MSP430 Architecture

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Introduction to MSP430

- MSP430 is a microcontroller family from Texas Instruments.
- It is one of the simplest microcontroller families from TI.
- ‘MSP’ stands for Mixed Signal Processor.
- It is a 16-bit processor (AB = 16 bit or 20 bit and DB =16 bit) designed for low power applications.
- It can be used for
 - General purpose sensing and measurement
 - Capacitive touch sensing
 - Ultrasonic sensing

Introduction to MSP430

- MSP430 series was launched in 1993 in Europe and in 1997 worldwide.
- The series has more than 500 different microcontrollers.
- MSP430 series support low power operation, over 5 low power modes are available.

MSP430 FAMILY

MSP430 SERIES	MSP430 FAMILY	FREQUENCY	MEMORY	GPIO
CAPACITIVE SENSING MCUs	FR25x/FR26x	16MHz	FRAM: UP TO 16KB SRAM: UP TO 4KB	15-19
VALUE LINE SENSING MCUs	FR2XX/FR4X	16MHz	FRAM: UP TO 16KB SRAM: UP TO 4 KB	16-64
	G2X/I2X	16MHz	FLASH: UP TO 56KB SRAM: UP TO 4KB	4-32
PERFORMANCE -SENSING MCUs	FR5X/FR6X	16MHz	FRAM: UP TO 256KB SRAM: UP TO 8KB	17-83
	F5X/F6X	25MHz	FLASH: 512KB SRAM: UP TO 67 KB	29-90
OTHER MSP430 MCUs	F2X/F4X	16MHz	FLASH: UP TO 120KB SRAM: UP TO 8KB	14-80
	F1X	16MHz	FLASH: UP TO 120KB SRAM: UP TO 10KB	10-48

MSP430 Nomenclature

MSP430F2618ATZQWT-EP

Processor Family

CC = Embedded RF Radio
MSP = Mixed Signal Processor
XMS = Experimental Silicon

430 MCU Platform

Device Type

Memory Type	Specialized Application
C = ROM	FG = Flash Medical
F = Flash	CG = ROM Medical
FR = FRAM	FE = Flash Energy Meter
G = Flash (Value Line)	FW = Flash Electronic Flow Meters
L = No non-volatile memory	AFE = Analog Front End
	BT = Pre-programmed with Bluetooth
	BQ = Contactless Power

Generation

1 series = up to 8 MHz
2 series = up to 16 MHz
3 series = Legacy OTP
4 series = up to 16 MHz w/ LCD

5 series = up to 25 MHz
6 series = up to 25 MHz w/ LCD
0 = Low Voltage Series

Family

Series and Device Number

Optional: A = revision

Optional: Temperature range

S = 0°C to 50°C
I = -40°C to 85°C
T = -40°C to 105°C

Packaging

www.ti.com/packaging

Optional: Distribution format

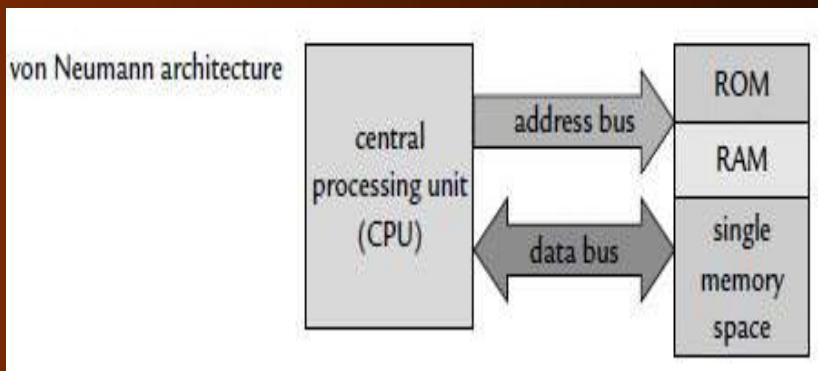
T = Small Reel (7-in)
R = Large Reel (11-in)
No markings = Tube or Tray

Optional: Additional features

*-Q1 = Automotive qualified
*-EP = Enhanced product (-40°C to 125°C)
*-HT = Extreme temp parts (-55°C to 150°C)

MSP430G2x53 Features

- 16-BIT RISC Architecture
- Von Neumann Memory Architecture



MSP430G2x53 Features

- Low Supply-Voltage Range: 1.8 V to 3.6 V
- Ultra-Low Power Consumption
 - Active Mode: 230 μ A at 1 MHz, 2.2 V
 - Standby Mode (LPM): 0.5 μ A
 - Off Mode (RAM Retention): 0.1 μ A
- Five Power Saving Modes (LPM0 -LPM4)
- Ultra-Fast Wake-Up From Standby Mode in Less Than 1 μ s

MSP430G2x53 Features

- **62.5ns Instruction Cycle Time (register to register operation)**
- **Basic Clock Module Configurations**
 - Internal Frequencies up to 16 MHz with four calibrated frequencies - 1MHz, 8Mhz, 12Mhz, 16Mhz.
 - Internal Very-Low-Power Low Frequency (LF) Oscillator.
 - 32-kHz Crystal as external clock source
 - External Digital Clock Source

MSP430G2x53 Features

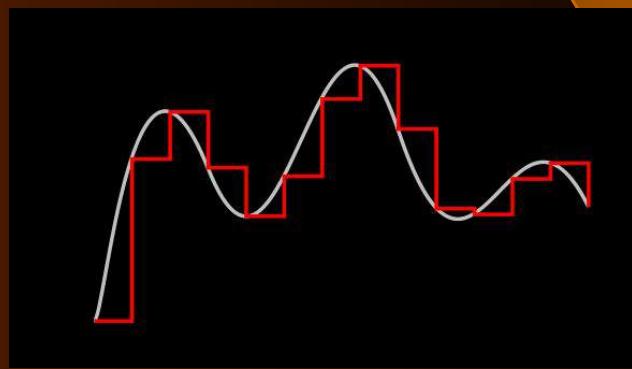
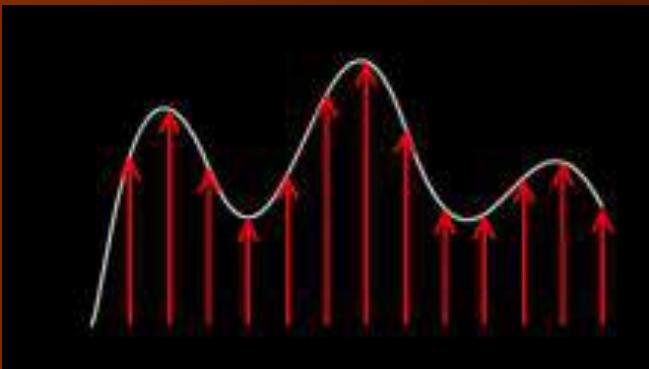
- **Two 16-Bit Timer_A With Three Capture/Compare Registers**
 - A timer is a hardware counter. Since 16 bit it counts from 0 to 65535.
- **Up to 24 Capacitive-Touch Enabled I/O Pins**
 - Inbuilt capacitive touch feature on all GPIO Pins

MSP430G2x53 Features

- **Universal Serial Communication Interface (USCI)**
 - UART
 - IrDA Encoder and Decoder
 - Synchronous SPI (Serial Peripheral Interface)
 - I2C (Inter IC Communication)
- **On-Chip Comparator**
- **Analog-to-Digital Cycle Time (A/D) Conversion**

MSP430G2x53 Features

- 10-Bit 200-ksps Analog-to-Digital (A/D) Converter
 - Internal Reference(1.5 V or 2.5 V)
 - Sample and Hold with programmable sample periods
 - Eight External Input channels



MSP430G2x53 Features

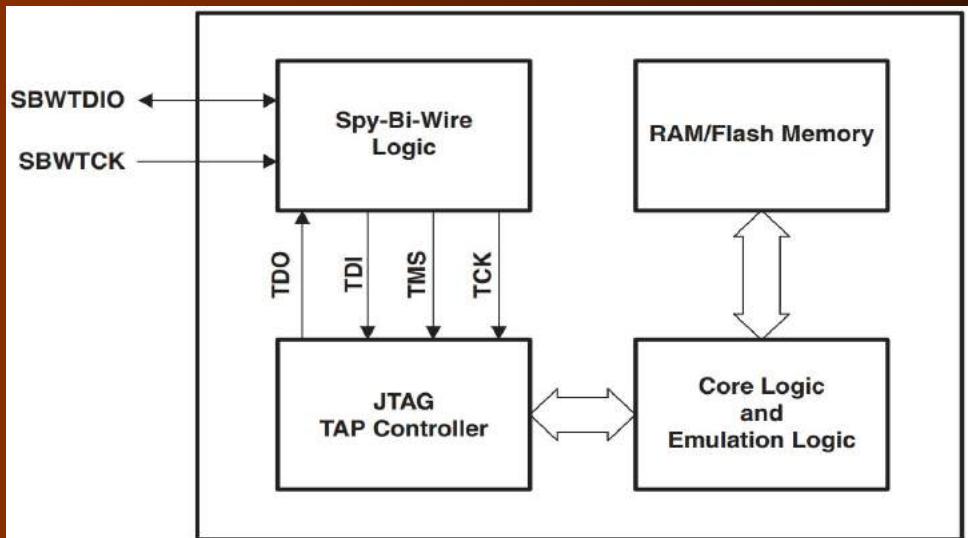
- Brownout Detector
- Serial Onboard Programming
- No External Programming Voltage Needed
- Programmable Code Protection by Security Fuse

MSP430G2x53 Features

- **On-Chip Emulation Logic With Spy-Bi-Wire**

Spy-Bi-Wire is a serialised JTAG protocol.

The two connections are a bidirectional data output (SBWTDO), and a clock (SBWTCK). The clocking signal is split into a period of three clock pulses, for each clock pulse the TDI, TDO and TMS signals are passed on the microcontroller via the bidirectional data output.



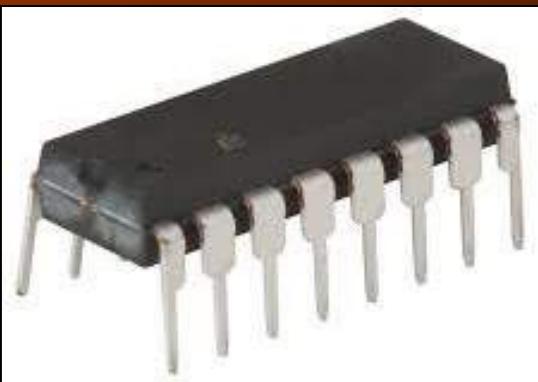
Spy-Bi-Wire Basic
Concept Diagram

MSP430G2x53 Features

- The SBWTCK signal is the clock signal and is a dedicated pin. In normal operation, this pin is internally pulled to ground. The SBWTDIO signal represents the data and is a bidirectional connection. To reduce the overhead of the 2-wire interface, the SBWTDIO line is shared with the RST/NMI pin of the device.

MSP430 Package Options

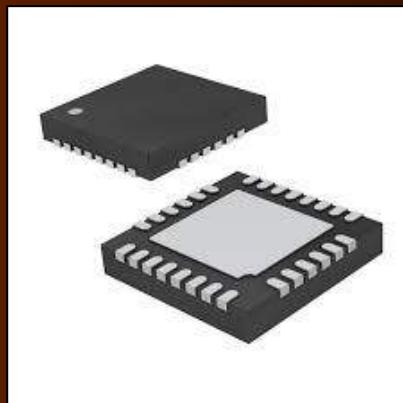
PDIP(PLASTIC DUAL IN-LINE PACKAGE)



TSSOP(THIN SHRINK SMALL OUTLINE PACKAGE)

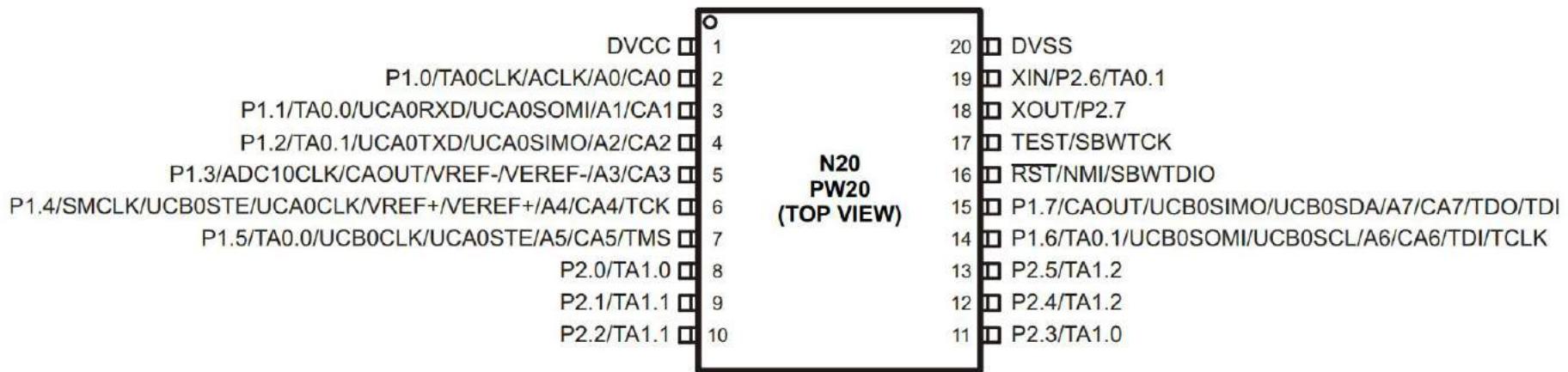


QFN(QUAD FLAT NO-LEAD)



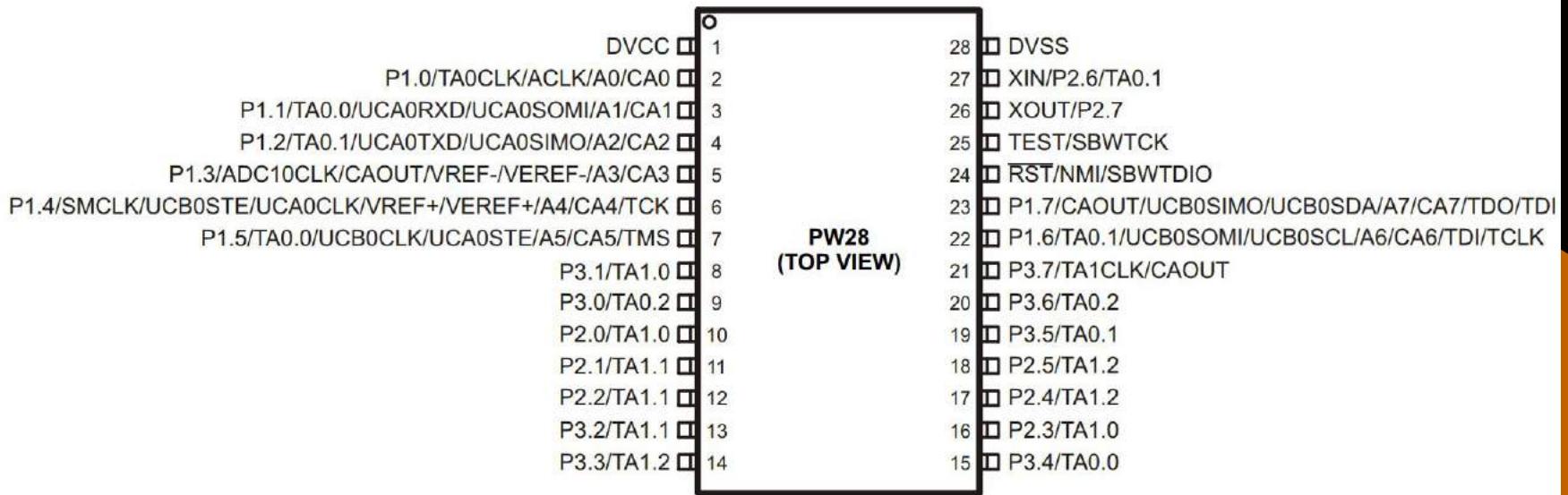
MSP430 PINOUT

Device Pinout, MSP430G2x13 and MSP430G2x53, 20-Pin Devices, TSSOP and PDIP



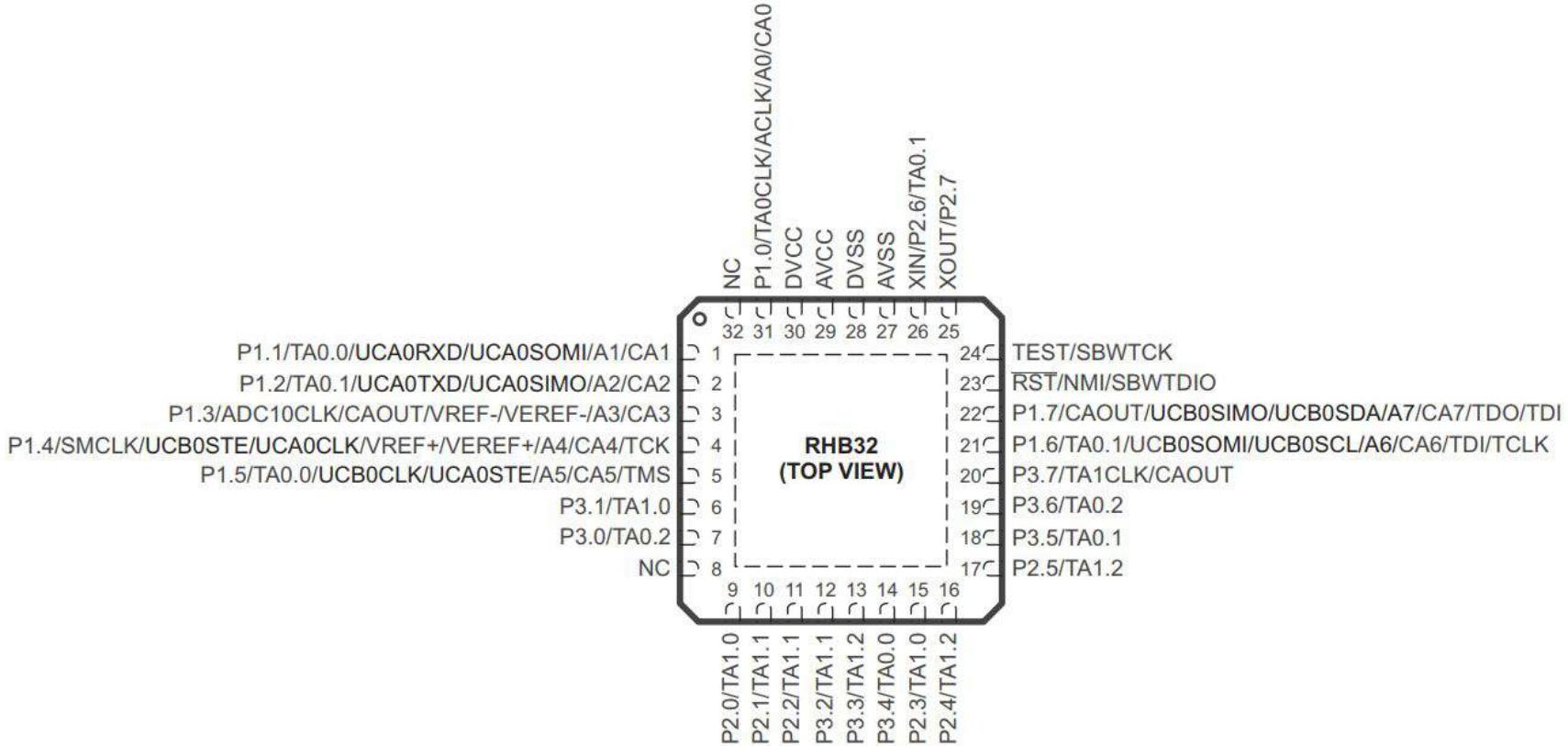
MSP430 PINOUT

Device Pinout, MSP430G2x13 and MSP430G2x53, 28-Pin Devices, TSSOP



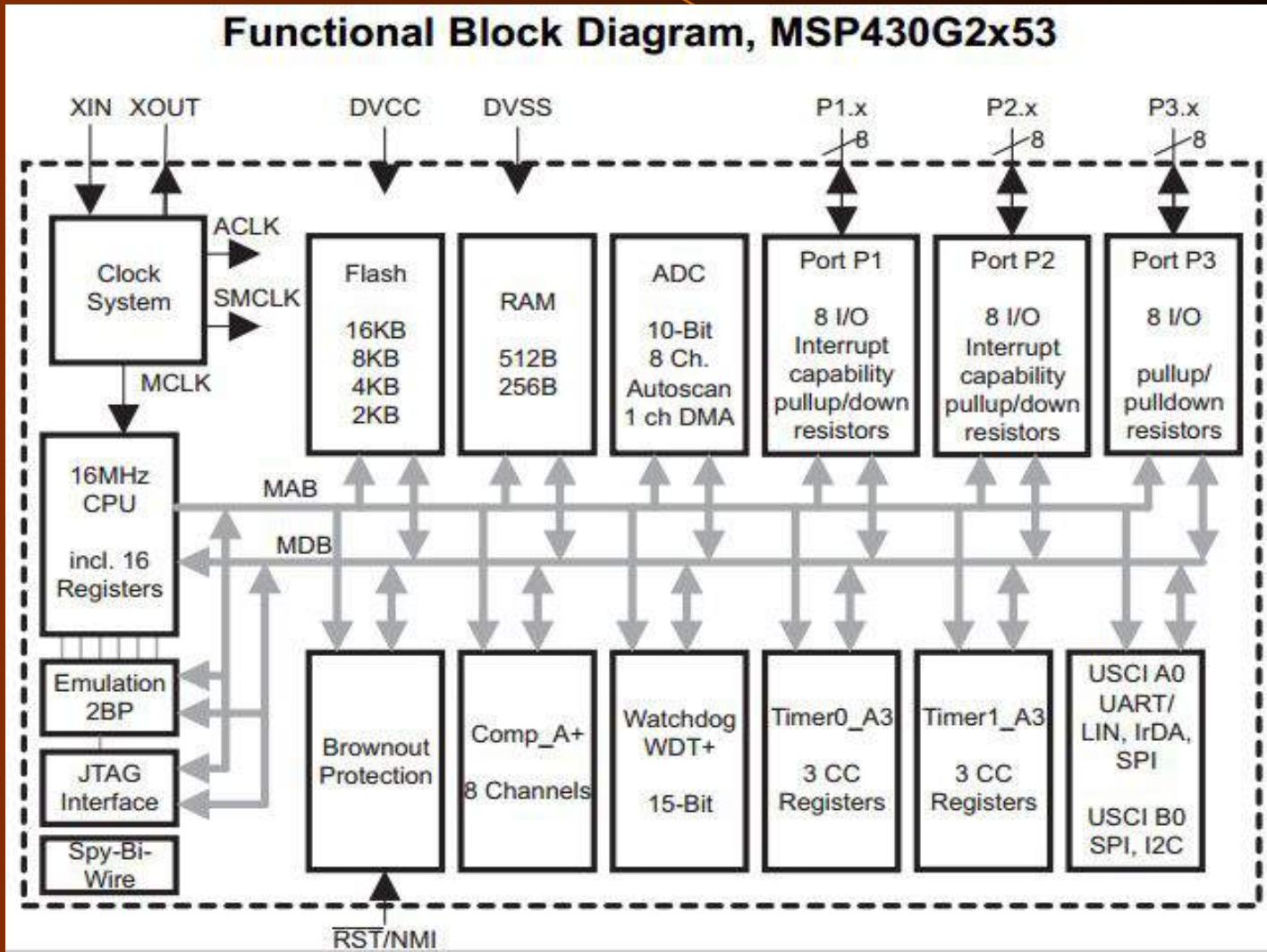
MSP430 PINOUT

Device Pinout, MSP430G2x13 and MSP430G2x53, 32-Pin Devices, QFN



NOTE: ADC10 is available on MSP430G2x53 devices only.

Functional Block Diagram



How to read the datasheet of the microcontroller

MSP430 Architecture

Introduction to the **MSP430 CPU**

- RISC Architecture with 27 core instructions and 7 addressing modes.
- 16-Bit Address Bus and 16-Bit Data Bus
- A set of 16 16-Bit Registers reduces fetches to memory
- Maximum clock frequency:- 16MHz(G-Series)
25MHz(F-Series)
- Constant generator provides six most used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding.
- Word and byte addressing and instruction formats.

Memory Architecture

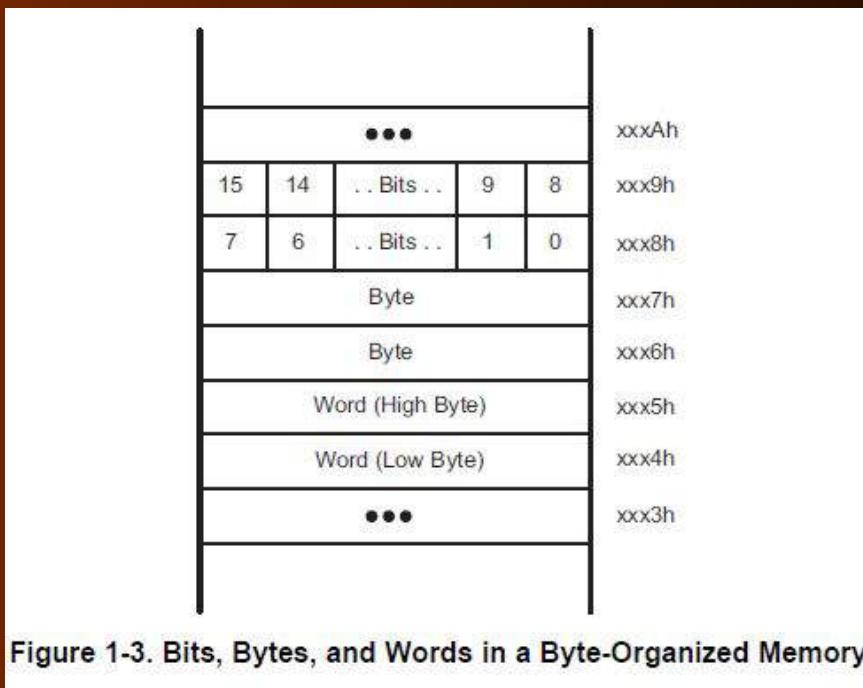
- MSP430 has Von Neumann Memory Architecture.
In von Neumann architecture there is only one set of addresses which cover both volatile and non-volatile memories.
- Address Bus is 16 bits wide, so $2^{16} = 65,536 = 64K$ addresses. So, Address Range is 0x0000 to 0xFFFF.
- MSP430X has an Address Bus of 20 Bits, so $2^{20} = 10,48,576$ addresses are possible.
- The memory data bus is 16 bits wide and can transfer either a word of 16 bits or a byte of 8 bits.

Memory Architecture

- The address of a word is defined to be the address of the byte with the lower address, which must be even. Eg: the two bytes at 0x4000 and 0x4001 can be fetched as a word with address 0x4000 in a single cycle.
- Instructions are composed of words and therefore must lie at even addresses.

Memory Architecture

- Words are stored as two bytes in the memory in the little endian ordering
- In little endian ordering, the lower order byte is stored at the lower address and the higher order byte at higher address.



MSP430 G Series Memory Map

		MSP430G2153	MSP430G2253 MSP430G2213	MSP430G2353 MSP430G2313	MSP430G2453 MSP430G2413	MSP430G2553 MSP430G2513
Memory	Size	1kB	2kB	4kB	8kB	16kB
Main: interrupt vector	Flash	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0
Main: code memory	Flash	0xFFFF to 0xFC00	0xFFFF to 0xF800	0xFFFF to 0xF000	0xFFFF to 0xE000	0xFFFF to 0xC000
Information memory	Size	256 Byte	256 Byte	256 Byte	256 Byte	256 Byte
	Flash	010FFh to 01000h	010FFh to 01000h	010FFh to 01000h	010FFh to 01000h	010FFh to 01000h
RAM	Size	256 Byte	256 Byte	256 Byte	512 Byte	512 Byte
		0x02FF to 0x0200	0x02FF to 0x0200	0x02FF to 0x0200	0x03FF to 0x0200	0x03FF to 0x0200
Peripherals	16-bit	01FFh to 0100h	01FFh to 0100h	01FFh to 0100h	01FFh to 0100h	01FFh to 0100h
	8-bit	0FFh to 010h	0FFh to 010h	0FFh to 010h	0FFh to 010h	0FFh to 010h
	8-bit SFR	0Fh to 00h	0Fh to 00h	0Fh to 00h	0Fh to 00h	0Fh to 00h

Memory Organization (Flash)

- MSP430 flash memory is partitioned into segments. A segment is the smallest size of flash memory that can be erased.
- The flash memory is partitioned into main and information memory sections.
- The differences between the two sections are the segment size and the physical addresses. The information memory has four 64-byte segments. The main memory has one or more 512-byte segments.

Memory Organization (Flash)

- The start address of Flash/ROM depends on the amount of Flash/ROM present and varies by device.
- The end address for Flash/ROM is 0xFFFF for devices with less than 60KB Flash/ROM.
- Flash can be used for both code and data but is generally used as code memory. The code memory holds the program.
- The interrupt vectors are used to handle the interrupts. The interrupt vector table is mapped into the upper 16 words of Flash/ROM address space, with the highest priority interrupt vector at the highest Flash/ROM word address (0xFFFFE).

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	Flash Code Memory (lower boundary varies)
0xFFB0	
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	peripheral registers
0x0100	(word access)
0x00FF	peripheral registers
0x0100	(byte access)
0x000F	special function registers
0x0000	(byte access)

8.4 Interrupt Vector Addresses

The interrupt vectors and the power-up starting address are located in the address range 0FFFFh to OFFC0h. The vector contains the 16-bit address of the appropriate interrupt handler instruction sequence.

If the reset vector (located at address 0FFEh) contains 0FFFFh (for example, flash is not programmed), the CPU goes into LPM4 immediately after power-up.

Table 5. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range ⁽¹⁾	PORIFG RSTIFG WDTIFG KEYV ⁽²⁾	Reset	0FFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG ⁽²⁾⁽³⁾	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	30
Timer1_A3	TA1CCR0 CCIFG ⁽⁴⁾	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG ⁽²⁾⁽⁴⁾	maskable	0FFF8h	28
Comparator_A+	CAIFG ⁽⁴⁾	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG ⁽⁴⁾	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG ⁽⁵⁾⁽⁴⁾	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 I ² C status	UCA0RXIFG, UCB0RXIFG ⁽²⁾⁽⁵⁾	maskable	0FFEEh	23
USCI_A0/USCI_B0 transmit USCI_B0 I ² C receive/transmit	UCA0TXIFG, UCB0TXIFG ⁽²⁾⁽⁶⁾	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG ⁽⁴⁾	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See ⁽⁷⁾			0FFDEh	15
See ⁽⁸⁾			0FFDEh to 0FFC0h	14 to 0, lowest

(1) A reset is generated if the CPU tries to fetch instructions from within the module register memory address range (0h to 01FFh) or from within unused address ranges.

(2) Multiple source flags

(3) (non)-maskable: the individual interrupt-enable bit can disable an interrupt event, but the general interrupt enable cannot.

(4) Interrupt flags are located in the module.

(5) In SPI mode: UCB0RXIFG. In I²C mode: UCALIFG, UCNACKIFG, ICSTTIFG, UCSTPIFG.

(6) In UART or SPI mode: UCB0TXIFG. In I²C mode: UCB0RXIFG, UCB0TXIFG.

(7) This location is used as bootstrap loader security key (BSLSKEY). A 0xAA55 at this location disables the BSL completely. A zero (0h) disables the erasure of the flash if an invalid password is supplied.

(8) The interrupt vectors at addresses 0FFDEh to 0FFC0h are not used in this device and can be used for regular program code if necessary.

Memory Organization (Information Memory)

- Information memory is a 256B block of flash memory that is intended for storage of nonvolatile data.
- Memory address range is: 0x1000h to 0x10FFh.
- The information memory has 4 segments of 64 bytes each. Segment A contains factory calibration data for the DCO in the MSP430G2553.
- After reset, segment A is protected against programming and erasing.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	peripheral registers
0x0100	(word access)
0x00FF	peripheral registers
0x0100	(byte access)
0x000F	special function registers
0x0000	(byte access)

Memory Organization (Information Memory)

- Bootstrap Loader memory is a 1023B block of flash memory that is intended for storage of nonvolatile data.
- Memory address range is: 0x0C00h to 0xFFFFh.
- Bootstrap Loader (BSL) memory space contains code which is invoked when the controllers enters into BSL programming mode

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	
0x0C00	Bootstrap Loader (BSL)
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	peripheral registers (word access)
0x0100	
0x00FF	peripheral registers (byte access)
0x0100	
0x000F	special function registers (byte access)
0x0000	

Memory Organization (RAM)

- RAM is used for data/variables.
- RAM starts at 0200h. The end address of RAM depends on the amount of RAM present and varies by device.
- MSP430G2553 has 512 Bytes of RAM.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	peripheral registers (word access)
0x0100	
0x00FF	peripheral registers (byte access)
0x0100	
0x000F	special function registers (byte access)
0x0000	

Memory Organization (Peripheral Modules)

- Peripheral registers are used by CPU to access and configure peripherals.
- The address space from 0100 to 01FFh is reserved for 16-bit peripheral modules. These modules should be accessed with word instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always 0.
- You can find the details of each peripheral register and its address in the ‘Peripheral File Map’ in the datasheet.

0xFFFF	Interrupt and Reset Vector Table
0xFFCO	
0xFFBF	
0xF000	Flash Code Memory (lower boundary varies)
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	peripheral registers (word access)
0x0100	
0x00FF	peripheral registers (byte access)
0x0100	
0x000F	special function registers (byte access)
0x0000	

Table 14. Peripherals With Word Access

MODULE	REGISTER DESCRIPTION	REGISTER NAME	OFFSET
ADC10 (MSP430G2x53 devices only)	ADC data transfer start address	ADC10SA	1BCh
	ADC memory	ADC10MEM	1B4h
	ADC control register 1	ADC10CTL1	1B2h
	ADC control register 0	ADC10CTL0	1B0h
Timer1_A3	Capture/compare register	TA1CCR2	0196h
	Capture/compare register	TA1CCR1	0194h
	Capture/compare register	TA1CCR0	0192h
	Timer_A register	TA1R	0190h
	Capture/compare control	TA1CCTL2	0186h
	Capture/compare control	TA1CCTL1	0184h
	Capture/compare control	TA1CCTL0	0182h
	Timer_A control	TA1CTL	0180h
	Timer_A interrupt vector	TA1IV	011Eh
Timer0_A3	Capture/compare register	TA0CCR2	0176h
	Capture/compare register	TA0CCR1	0174h
	Capture/compare register	TA0CCR0	0172h
	Timer_A register	TA0R	0170h
	Capture/compare control	TA0CCTL2	0166h
	Capture/compare control	TA0CCTL1	0164h
	Capture/compare control	TA0CCTL0	0162h
	Timer_A control	TA0CTL	0160h
	Timer_A interrupt vector	TA0IV	012Eh
Flash Memory	Flash control 3	FCTL3	012Ch
	Flash control 2	FCTL2	012Ah
	Flash control 1	FCTL1	0128h
Watchdog Timer+	Watchdog/timer control	WDTCTL	0120h

Memory Organization (Peripheral Modules)

- The address space from 010h to 0FFh is reserved for 8-bit peripheral modules. These modules should be accessed with byte instructions.
- Read access of byte modules using word instructions results in unpredictable data in the high byte. If word data is written to a byte module only the low byte is written into the peripheral register, ignoring the high byte.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	
0xF000	Flash Code Memory (lower boundary varies)
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM
0x0200	(upper boundary varies)
0x01FF	peripheral registers
0x0100	(word access)
0x00FF	peripheral registers
0x0100	(byte access)
0x000F	special function registers
0x0000	(byte access)

Table 15. Peripherals With Byte Access

MODULE	REGISTER DESCRIPTION	REGISTER NAME	OFFSET
USCI_B0	USCI_B0 transmit buffer	UCB0TXBUF	06Fh
	USCI_B0 receive buffer	UCB0RXBUF	06Eh
	USCI_B0 status	UCB0STAT	06Dh
	USCI_B0 I2C interrupt enable	UCB0CIE	06Ch
	USCI_B0 bit rate control 1	UCB0BR1	06Bh
	USCI_B0 bit rate control 0	UCB0BR0	06Ah
	USCI_B0 control 1	UCB0CTL1	069h
	USCI_B0 control 0	UCB0CTL0	068h
	USCI_B0 I2C slave address	UCB0SA	011Ah
	USCI_B0 I2C own address	UCB0OA	0118h
USCI_A0	USCI_A0 transmit buffer	UCA0TXBUF	067h
	USCI_A0 receive buffer	UCA0RXBUF	066h
	USCI_A0 status	UCA0STAT	065h
	USCI_A0 modulation control	UCA0MCTL	064h
	USCI_A0 baud rate control 1	UCA0BR1	063h
	USCI_A0 baud rate control 0	UCA0BR0	062h
	USCI_A0 control 1	UCA0CTL1	061h
	USCI_A0 control 0	UCA0CTL0	060h
	USCI_A0 IrDA receive control	UCA0IRRCTL	05Fh
	USCI_A0 IrDA transmit control	UCA0IRTCTL	05Eh
ADC10 (MSP430G2x53 devices only)	ADC analog enable 0	ADC10AE0	04Ah
	ADC analog enable 1	ADC10AE1	04Bh
	ADC data transfer control register 1	ADC10DTC1	049h
	ADC data transfer control register 0	ADC10DTC0	048h
Comparator_A+	Comparator_A+ port disable	CAPD	05Bh
	Comparator_A+ control 2	CACTL2	05Ah
	Comparator_A+ control 1	CACTL1	059h
Basic Clock System+	Basic clock system control 3	BCSCTL3	053h
	Basic clock system control 2	BCSCTL2	058h
	Basic clock system control 1	BCSCTL1	057h
	DCO clock frequency control	DCOCTL	056h
Port P3 (28-pin PW and 32-pin RHB only)	Port P3 selection 2, pin	P3SEL2	043h
	Port P3 resistor enable	P3REN	010h
	Port P3 selection	P3SEL	01Bh
	Port P3 direction	P3DIR	01Ah
	Port P3 output	P3OUT	019h
	Port P3 input	P3IN	018h
Port P2	Port P2 selection 2	P2SEL2	042h
	Port P2 resistor enable	P2REN	02Fh
	Port P2 selection	P2SEL	02Eh
	Port P2 interrupt enable	P2IE	02Dh
	Port P2 interrupt edge select	P2IES	02Ch
	Port P2 interrupt flag	P2IFG	02Bh
	Port P2 direction	P2DIR	02Ah
	Port P2 output	P2OUT	029h
	Port P2 input	P2IN	028h

Table 15. Peripherals With Byte Access (continued)

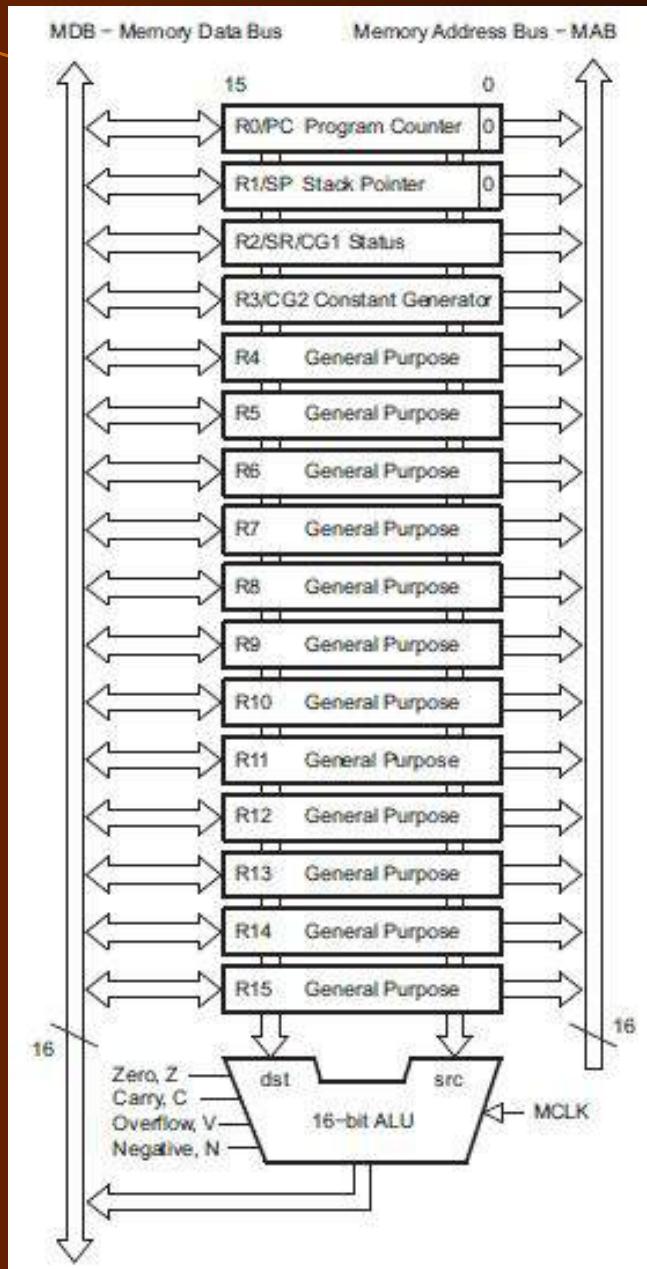
MODULE	REGISTER DESCRIPTION	REGISTER NAME	OFFSET
Port P1	Port P1 selection 2	P1SEL2	041h
	Port P1 resistor enable	P1REN	027h
	Port P1 selection	P1SEL	026h
	Port P1 interrupt enable	P1IE	025h
	Port P1 interrupt edge select	P1IES	024h
	Port P1 interrupt flag	P1IFG	023h
	Port P1 direction	P1DIR	022h
	Port P1 output	P1OUT	021h
	Port P1 input	P1IN	020h
Special Function	SFR interrupt flag 2	IFG2	003h
	SFR interrupt flag 1	IFG1	002h
	SFR interrupt enable 2	IE2	001h
	SFR interrupt enable 1	IE1	000h

Memory Organization (Special Function Registers)

- Some peripheral functions are configured in the SFRs. The SFRs are located in the lower 16 bytes of the address space, and are organized by byte. SFRs must be accessed using byte instructions only.
- MSP430G2553 has interrupt enables and interrupt flag registers as SFRs.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	peripheral registers
0x0100	(word access)
0x00FF	peripheral registers
0x0100	(byte access)
0x000F	special function registers
0x0000	(byte access)

MSP430 CPU



MSP430 CPU Block Diagram

CPU Registers

- 16 Registers: The generous set of 16 registers is characteristic of a RISC CPU.
- These Registers provide reduced instruction execution time.
- Register to register operation takes only clock cycle.
- 4 Special Purpose
 - Program Counter(PC)
 - Stack Pointer(SP)
 - Status Register(SR)
 - Constant Generator
- 12 General Purpose

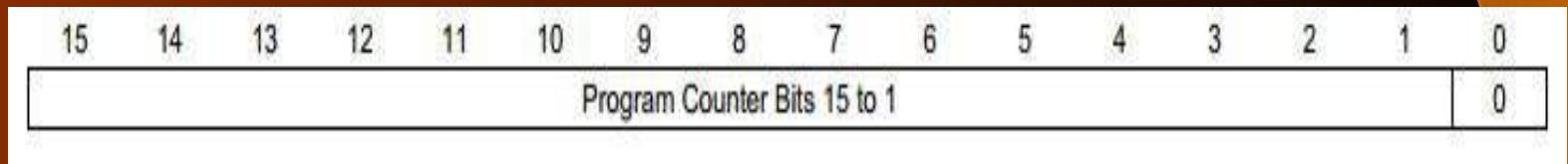
Program Counter	PC/R0
Stack Pointer	SP/R1
Status Register	SR/CG1/R2
Constant Generator	CG2/R3
General-Purpose Register	R4
General-Purpose Register	R5
General-Purpose Register	R6
General-Purpose Register	R7
General-Purpose Register	R8
General-Purpose Register	R9
General-Purpose Register	R10
General-Purpose Register	R11
General-Purpose Register	R12
General-Purpose Register	R13
General-Purpose Register	R14
General-Purpose Register	R15

CPU Registers

CPU Registers

Program Counter

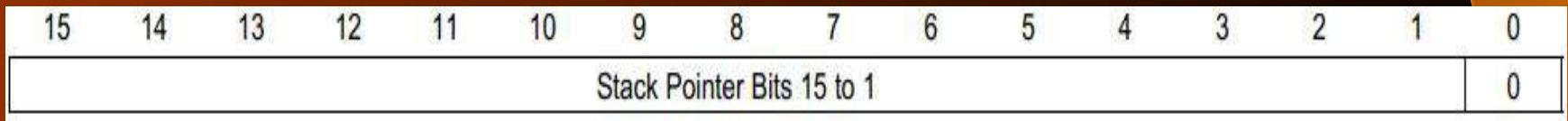
- This contains the address of the next instruction to be executed. Instructions are composed of 1–3 words, which must be aligned to even addresses, so the LSB of the PC is hard-wired to 0.



CPU Registers

Stack Pointer

- The stack pointer is used by the CPU to store the return addresses of subroutine calls and interrupts.
- The SP is initialized into RAM by the user, and is aligned to even addresses.
- SP hold the address of the most recently added word in the stack and is automatically adjusted as the stack goes up and down.



CPU Registers

Status Register

- This contains a set of flags (single bits), whose functions fall into three categories.
 - The most commonly used flags are C, Z, N, and V, which give information about the result of the last arithmetic or logical operation. Decisions that affect the flow of control in the program can be made by testing these bits.
 - Setting the GIE bit enables maskable interrupts.
 - The final group of bits is CPUOFF, OSCOFF, SCG0, and SCG1, which control the mode of operation of the MCU.

Status Register

Bit	Description	
V	Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	
	ADD (.B), ADDC (.B)	Set when: Positive + Positive = Negative Negative + Negative = Positive Otherwise reset
	SUB (.B), SUBC (.B), CMP (.B)	Set when: Positive – Negative = Negative Negative – Positive = Positive Otherwise reset
SCG1	System clock generator 1. When set, turns off the SMCLK.	
SCG0	System clock generator 0. When set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.	
OSCOFF	Oscillator Off. When set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK.	
CPUOFF	CPU off. When set, turns off the CPU.	
GIE	General interrupt enable. When set, enables maskable interrupts. When reset, all maskable interrupts are disabled.	
N	Negative bit. Set when the result of a byte or word operation is negative and cleared when the result is not negative. Word operation: N is set to the value of bit 15 of the result. Byte operation: N is set to the value of bit 7 of the result.	
Z	Zero bit. Set when the result of a byte or word operation is 0 and cleared when the result is not 0.	
C	Carry bit. Set when the result of a byte or word operation produced a carry and cleared when no carry occurred.	

CPU Registers

Constant Generator Registers

- Six commonly-used constants are generated with the constant generator registers R2 and R3, without requiring an additional 16-bit word of program code. The constants are selected with the source-register addressing modes (As).
- The constant generator advantages are:
 - No special instructions required
 - No additional code word for the six constants
 - No code memory access required to fetch the constant

CPU Registers

General Purpose Registers

- The remaining 12 registers R4–R15 have no dedicated purpose and may be used as general working registers.
- They may be used for either data or addresses because both are 16-bit values, which simplifies the operation significantly.

MSP430 Instruction Set Architecture

Instruction Format

There are three formats of instructions in the MSP430:

- **Double operand:** Arithmetic and logical operations with two operands such as ADD R4,R5. Both operands must be specified in the instruction. This contrasts with accumulator-based architectures, where an accumulator or working register is used automatically as the destination and one operand.
- **Single operand:** A mixture of instructions for control or to manipulate a single operand, which is effectively the *source* for the addressing modes.
- **Jumps:** The jump to the destination rather than its absolute address, in other words the offset that must be added to the program counter.

Table 3. Instruction Word Formats

INSTRUCTION FORMAT	EXAMPLE	OPERATION
Dual operands, source-destination	ADD R4,R5	R4 + R5 ---> R5
Single operands, destination only	CALL R8	PC -->(TOS), R8--> PC
Relative jump, un/conditional	JNE	Jump-on-equal bit = 0

Addressing Modes

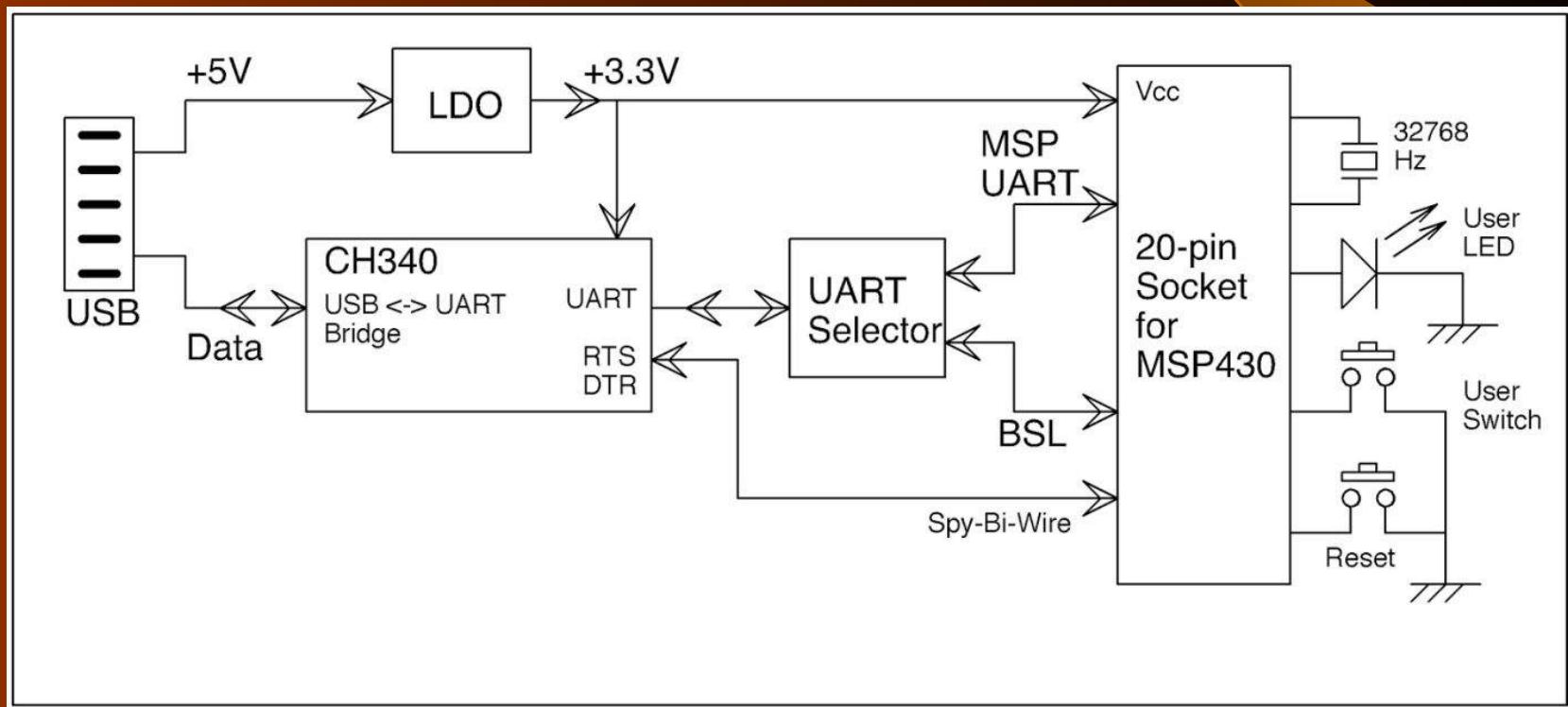
- Addressing modes are the ways in which operands can be specified.
- MSP430 has seven addressing modes. These are:
 1. Register Mode
 2. Indexed Mode
 3. Symbolic Mode
 4. Absolute Mode
 5. Indirect Register Mode
 6. Indirect Autoincrement Mode
 7. Immediate Mode

Source/Destination Operand Addressing Modes

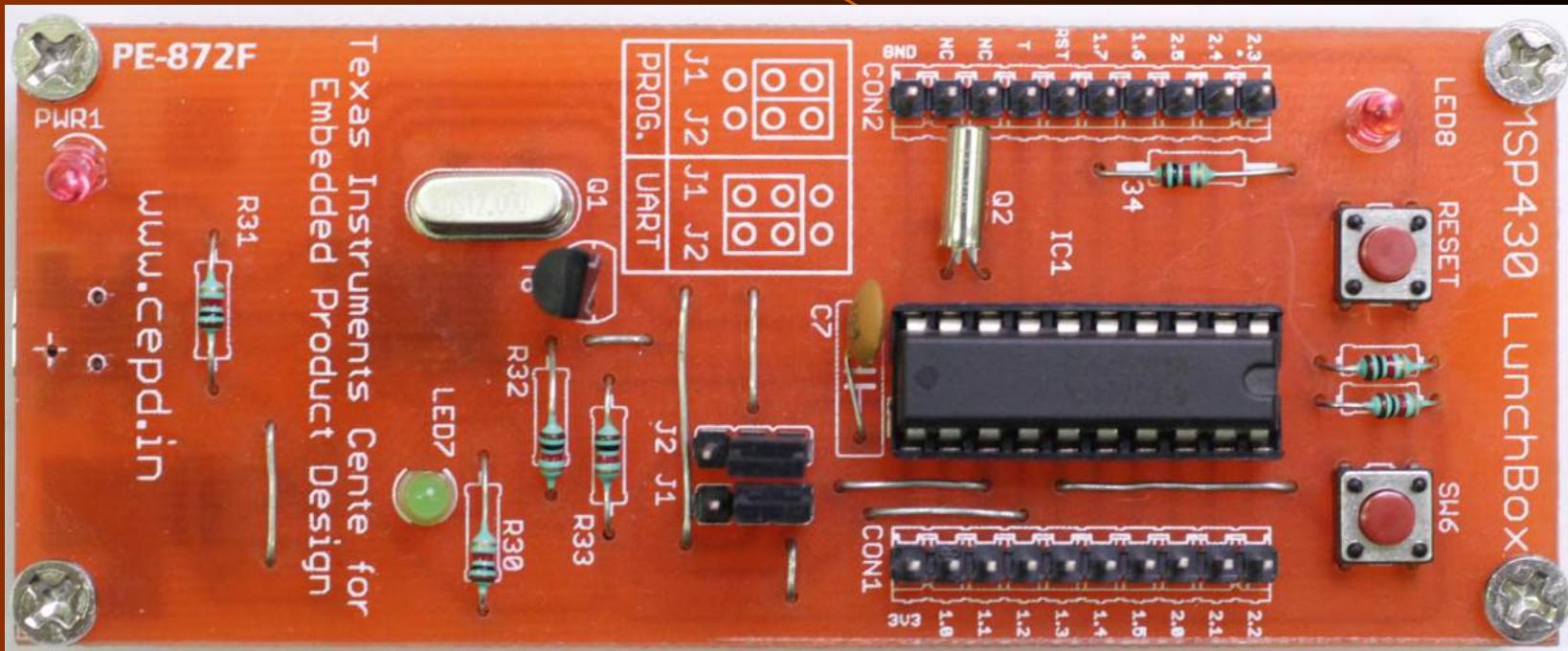
As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/-	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/-	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/-	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

Introduction to MSP430 Lunchbox

MSP430 LunchBox is a low cost DIY platform for learning microcontrollers and physical computing.



MSP430 Lunchbox



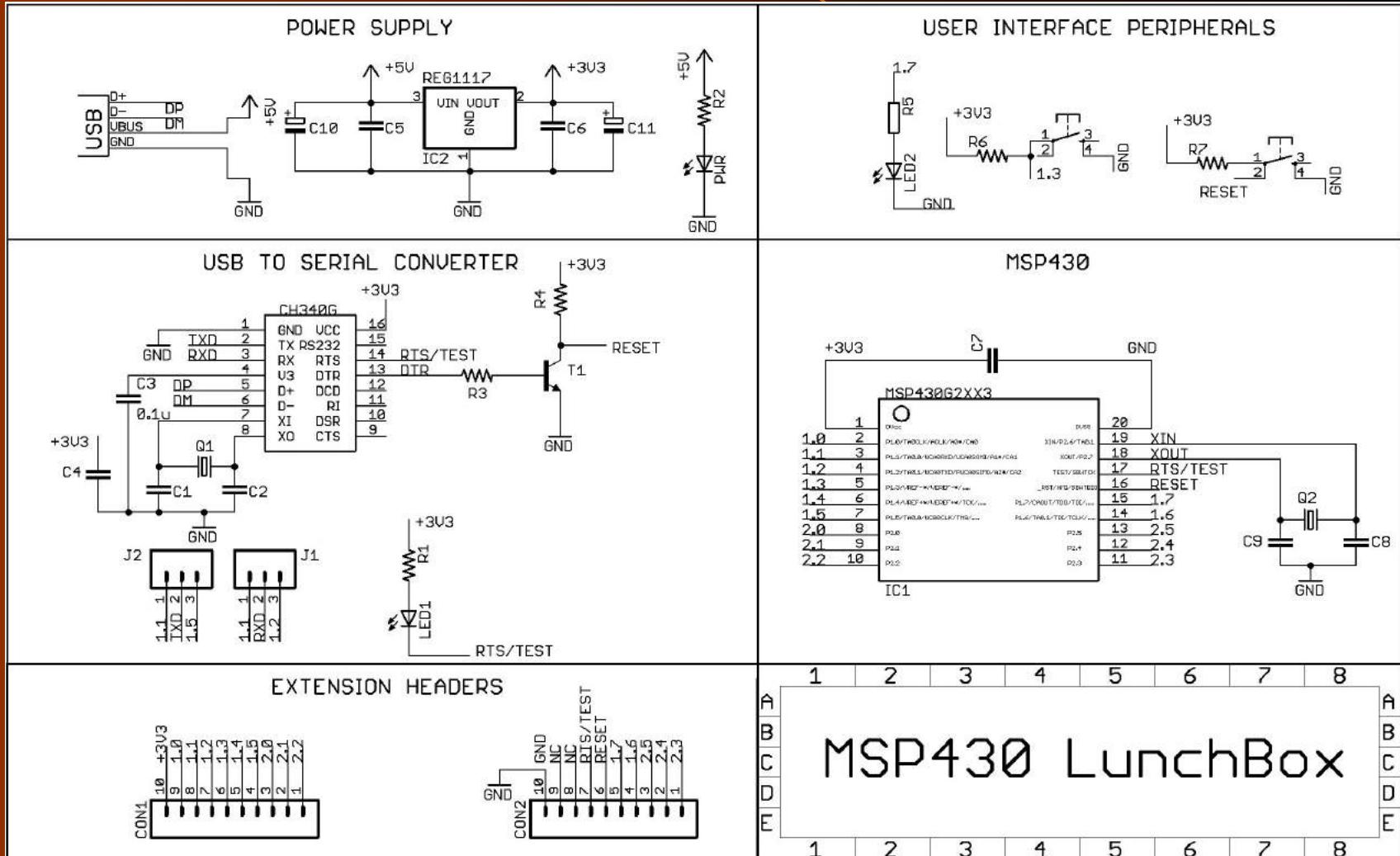
- Based on MSP430G2553 MCU
- On-board LED and push button for experimentation
- Programmable via TI Code Composer Studio

MSP430G2553

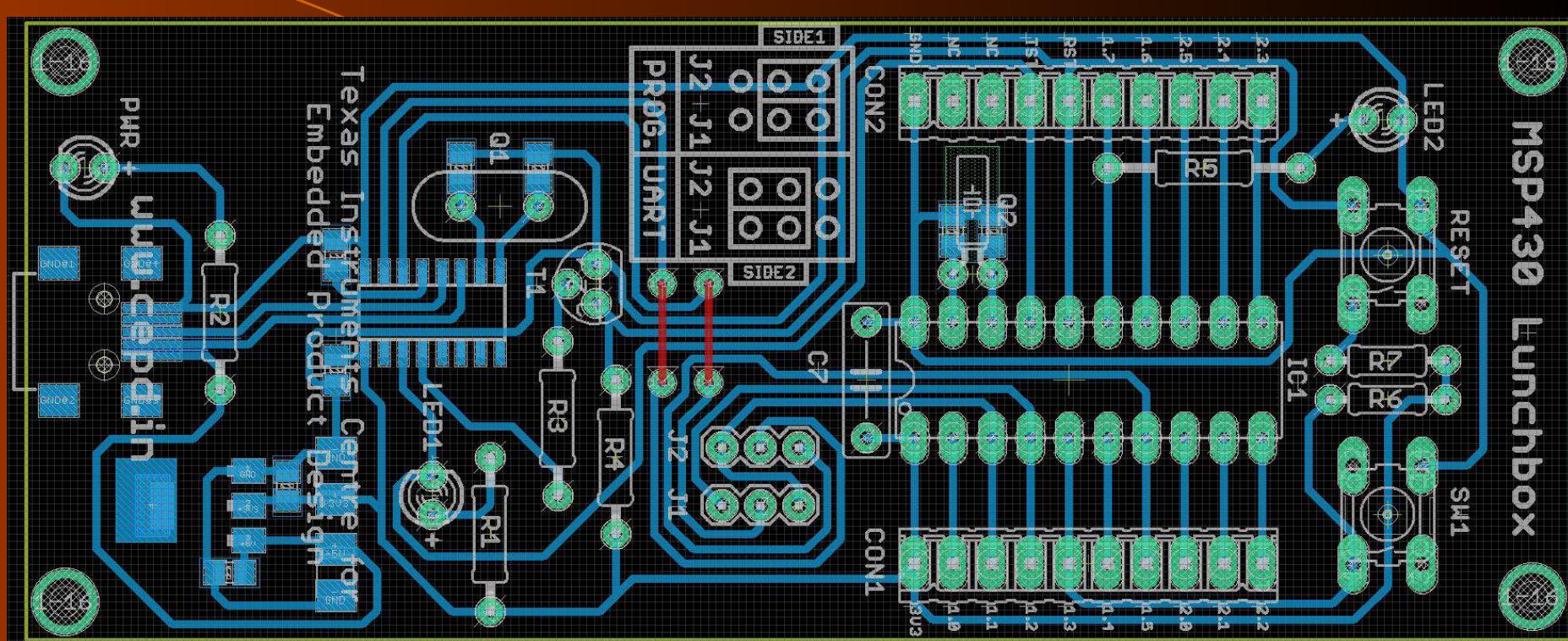


- 16 KB FLASH
- 512 BYTE RAM
- 16 GPIOs
- 8 CHANNEL ADC
- 2 TIMER_A3
- 8 CHANNEL COMPARATOR_A+

Lunchbox Schematic



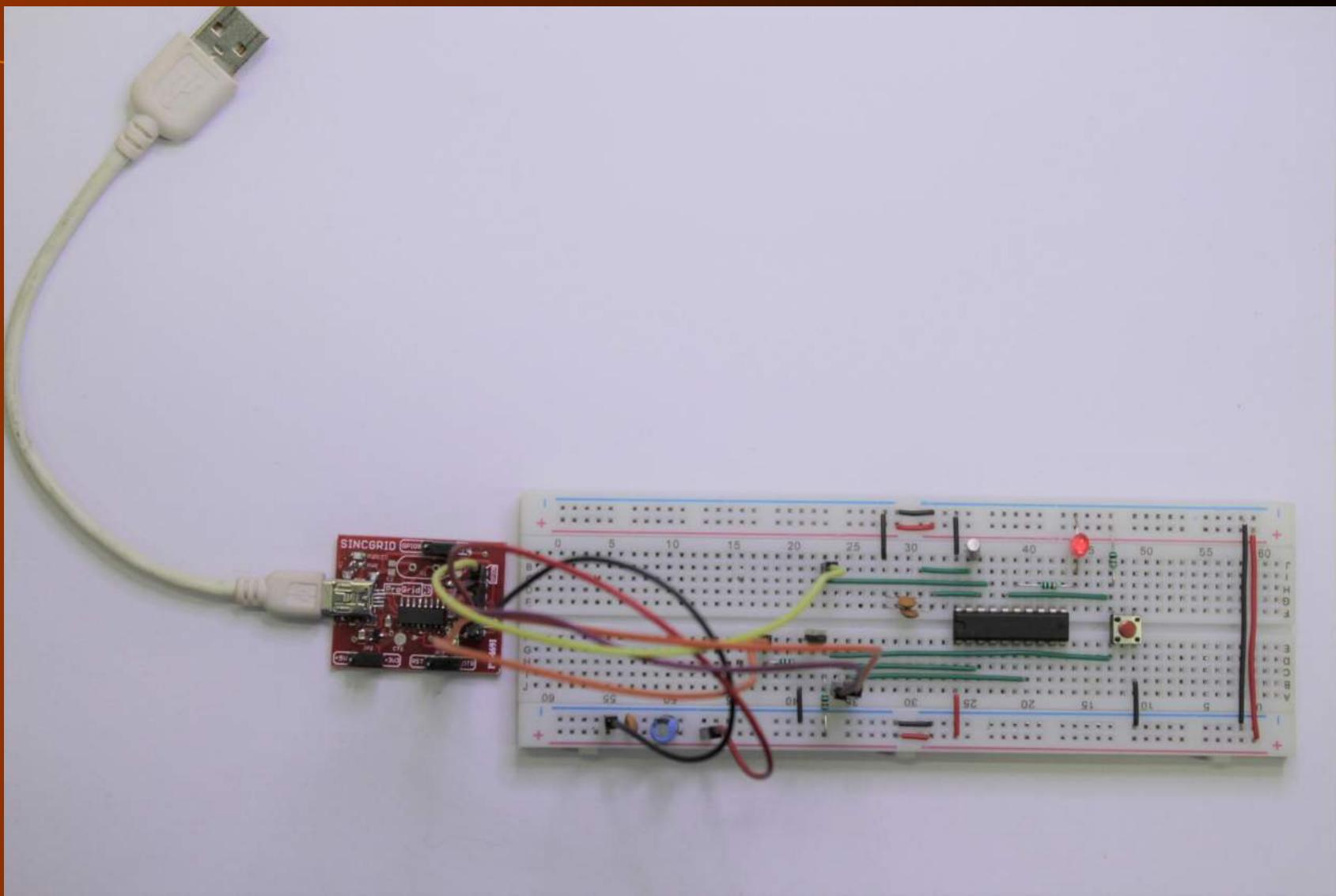
Lunchbox Board Layout



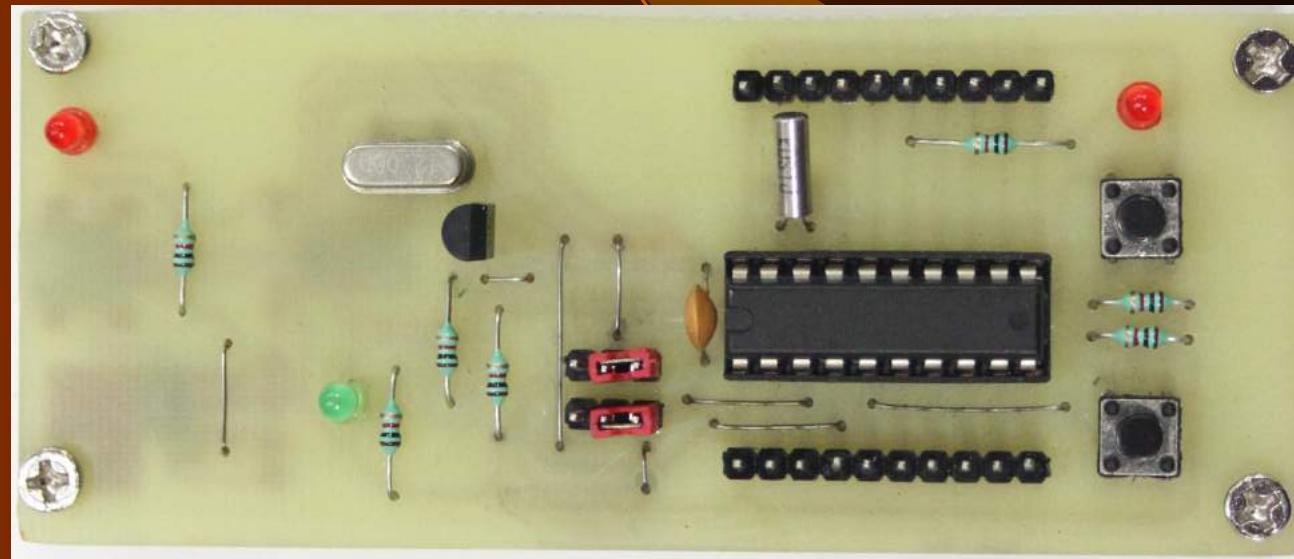
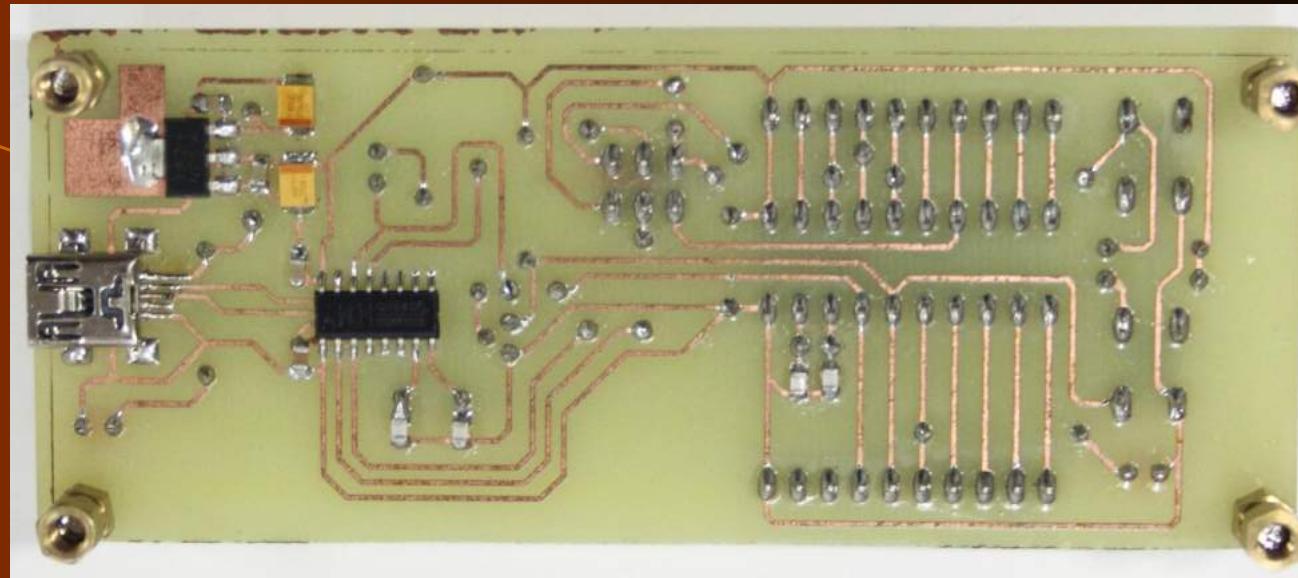
Lunchbox Development Process

- Lunchbox on a Breadboard/Zero Board
- Inhouse PCB
- Manufactured PCB

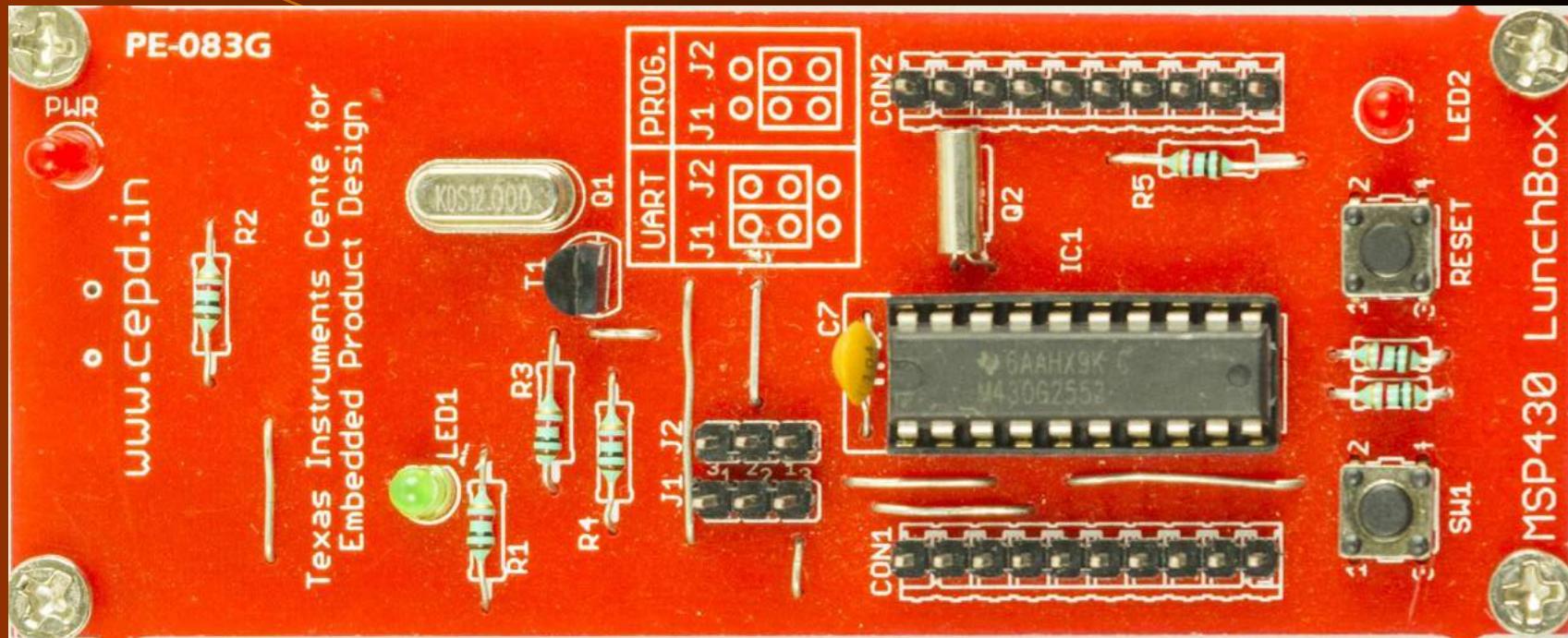
Lunchbox on BreadBoard



Lunchbox In-house PCB



Manufactured Lunchbox





Thank you!

Introduction to Embedded System Design

Programming the MSP430

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Software Development Tools for the MSP430

- Texas Instruments Code Composer Studio IDE
- Energia IDE
- Third Party Development Environments (IAR)
- GCC Open Source Packages

Code Composer Studio IDE

- Code Composer Studio (CCS) is the integrated development environment for TI's DSPs, microcontrollers and application processors. Code Composer Studio includes tools used to develop and debug embedded applications.
- It includes:-
 1. Compilers
 2. Source code editor
 3. Project build environment
 4. Debugger
 5. Simulators
- The intuitive IDE provides a single user interface taking you through each step of the application development flow.

Code Composer Studio IDE

- Code Composer Studio combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers.
- Supported on Mac OS, Windows, and Linux.
- E2E support forum from TI for resolving issues.

Code Composer Studio IDE

workspace_v7 - CCS Edit - gpiointerrupt_CC1310_LAUNCHXL_tirtos_ccs/targetConfigs/CC1310F128.cxml - Code Composer Studio

File Edit View Navigate Project Run Scripts Window Help

Project Explorer X gpiointerrupt_CC1310_LAUNCHXL_tirtos_ccs [Active - Del] Includes Debug targetConfigs CC1310F128.cxml [Active/Default] readme.txt Board.h CC1310_LAUNCHXL_TIRTOS.cmd CC1310_LAUNCHXL.c CC1310_LAUNCHXL.h ccfg.c gpiointerrupt.c main_tirtos.c Board.html README.html README.md tirtos_builds_CC1310_LAUNCHXL_release_ccs

Getting Started Resource Explorer CC1310F128.cxml X

Basic

General Setup
This section describes the general configuration about the target.

Connection: Texas Instruments XDS110 USB Debug Probe

Board or Device: type filter text

- AMIC110
- BeagleBoard
- BeagleBoard_xM
- BeagleBone
- BeagleBone_Black
- Blaze_4430
- Blaze_4460
- C6A8167
- C6A8168
- CC1310F128
- CC1310F64

SimpleLink(TM) Sub-1 GHz CC1310 wireless MCU

Note: Support for more devices may be available from the update manager.

Advanced Setup

Target Configuration: lists the configuration options for the selected target.

Save Configuration

Save

Test Connection

To test a connection, all changes must have been saved. The configuration file contains no errors and the connection can be tested.

Test Connection

Alternate Communication

Uart Communication

To enable host side (i.e. PC) configuration needs to be configured. Communication over UART, target application implementation. Please check example project for more information. If the target application leverages TI-RTOS, then please enable Uart Monitor module.

To add a port in the target application for Uart communication, click the Add Port button.

To remove a port in the target application for Uart communication, click the Remove button.

Basic Advanced Source

Energia IDE

- An open source & community-driven IDE and software framework. Based on the Wiring framework, Energia provides an intuitive coding environment as well as a robust framework of easy-to-use functional APIs & libraries for programming a microcontroller.
- Supported on Mac OS, Windows, and Linux.

Energia IDE

Energia_Rocks.ino | Energia 1.6.10E18

The screenshot shows the Energia IDE interface. The title bar reads "Energia_Rocks.ino | Energia 1.6.10E18". The main window contains the following code:

```
1 #define LED RED_LED
2
3 // the setup routine runs once when you press reset:
4 void setup() {
5     // initialize the digital pin as an output.
6     pinMode(LED, OUTPUT);
7 }
8
9 // the loop routine runs over and over again forever:
10 void loop() {
11     digitalWrite(LED, HIGH);    // turn the LED on (HIGH is the voltage level)
12     delay(1000);               // wait for a second
13     digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
14     delay(1000);               // wait for a second
15 }
```

Below the code editor is a preview window featuring a red circle with a white rocket ship launching from it, and the word "Energia" in large red letters.

Done Saving.

15 RED LaunchPad w/ msp432 EMT (48MHz) on /dev/cu.usbmodemM4321001

IAR Embedded Workbench

- A complete debugger and C/C++ compiler toolchain for building and debugging embedded applications based on MSP430 MCUs. The debugger is fully integrated for source with support for complex code and data breakpoints.

IAR Embedded Workbench

LCD - IAR Embedded Workbench IDE

File Edit View Project I-jet/JTAGjet Tools Window Help

Workspace

Flash Debug

Files

LCD - Flash De... ▾

- app
 - iar_logo.c
 - main.c
 - stm32f4xx_it.c
 - Terminal_18.c
 - Terminal_6.c
 - Terminal_9.c
- board
 - iar_stm32f40...
- modules
 - drv_glcd.c
 - glcd_ll.c
- startup
 - startup_stm...
 - system_stm...
- STM32F4xx_St...
- readme.txt
- Output

LCD

Information Center for ARM main *

```
        }
    else
    {
        GLCD_print("\fBacklight adj.\r");
    }

    static int loopCounter = 0;

    while(1)
    {
        loopCounter++;

        DelayResolution100us(1000);

        STM_LEDOff((Led_TypeDef) (cntr & 0x03));

        STM_LEDOn((Led_TypeDef) (++cntr & 0x03));

        if(Bit_SET == STM_ButtonGetState(BUTTON_RIGHT))
        {
            CntrSel = TRUE;
            GLCD_print("\fContrast adj.\r");
        }
    }
}
```

Messages

- system_stm32f4xx.c
- Terminal_18_24x12.c
- Terminal_6_8x6.c
- Terminal_9_12x6.c
- Linking

Total number of errors: 0
Total number of warnings: 0

Build

Debug Log Build

Download the application and start the debugger

Errors 0, Warnings 0 Ln 235, Col 5 System NUM



GCC Open Source Packages

- Free, complete debugger and open source C/C++ compiler toolchains for building and debugging embedded applications using MSP430 MCUs without code size limitations. The compilers can be used standalone from the command-line or used within the Code Composer Studio IDE.

Program Download Mechanisms on the MSP430

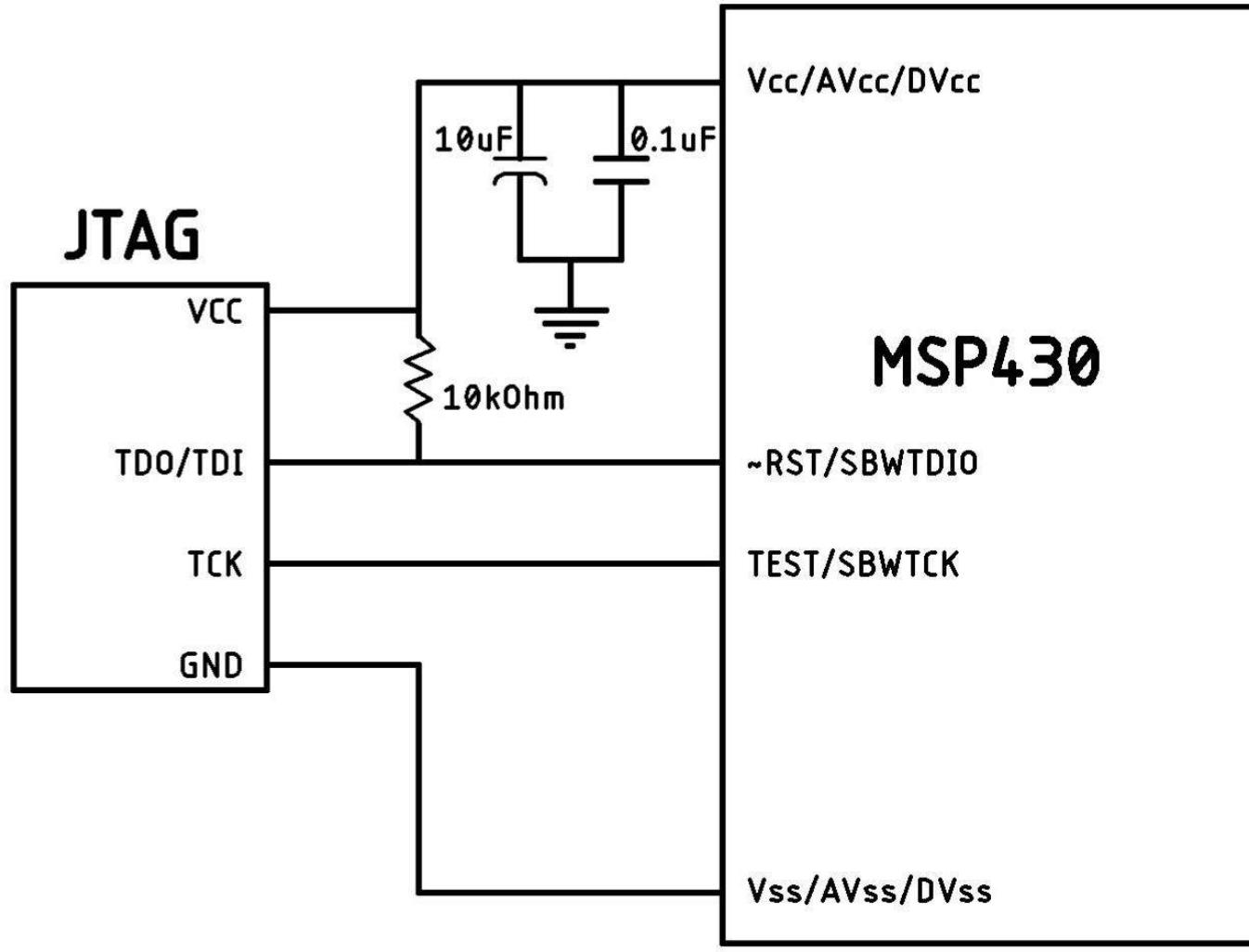
- Spy-Bi-Wire (SBW) using TEST/RESET
- JTAG Debugger
- UART/USB Bootloaders

SPY-BI-WIRE

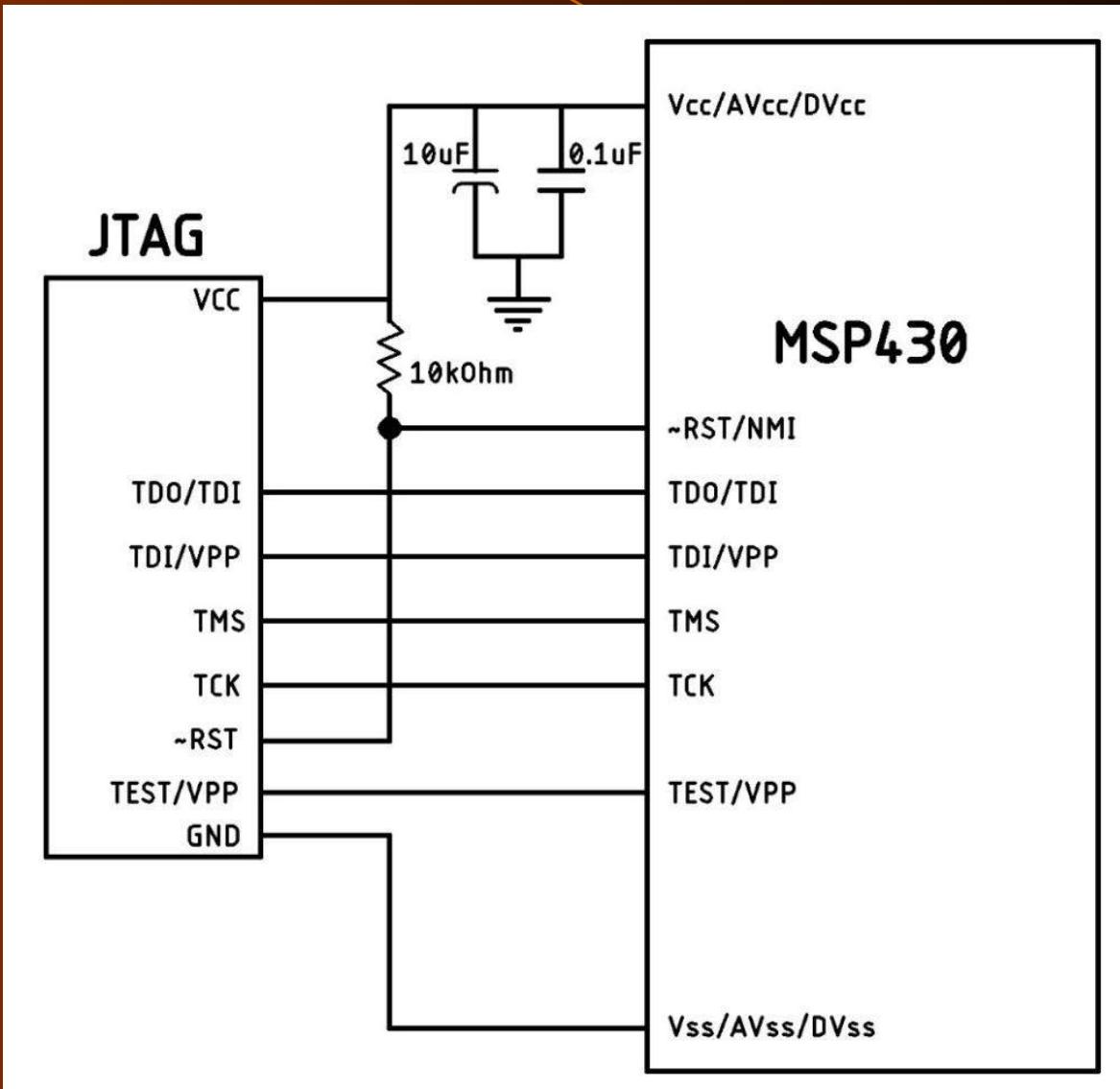
- Also known as 2 wire JTAG

PROS	CONS
<ul style="list-style-type: none">● Only 2 wires used (TEST & RESET)● No overlapping with GPIO	<ul style="list-style-type: none">● Slower than 4 wire JTAG

SPY-BI-WIRE



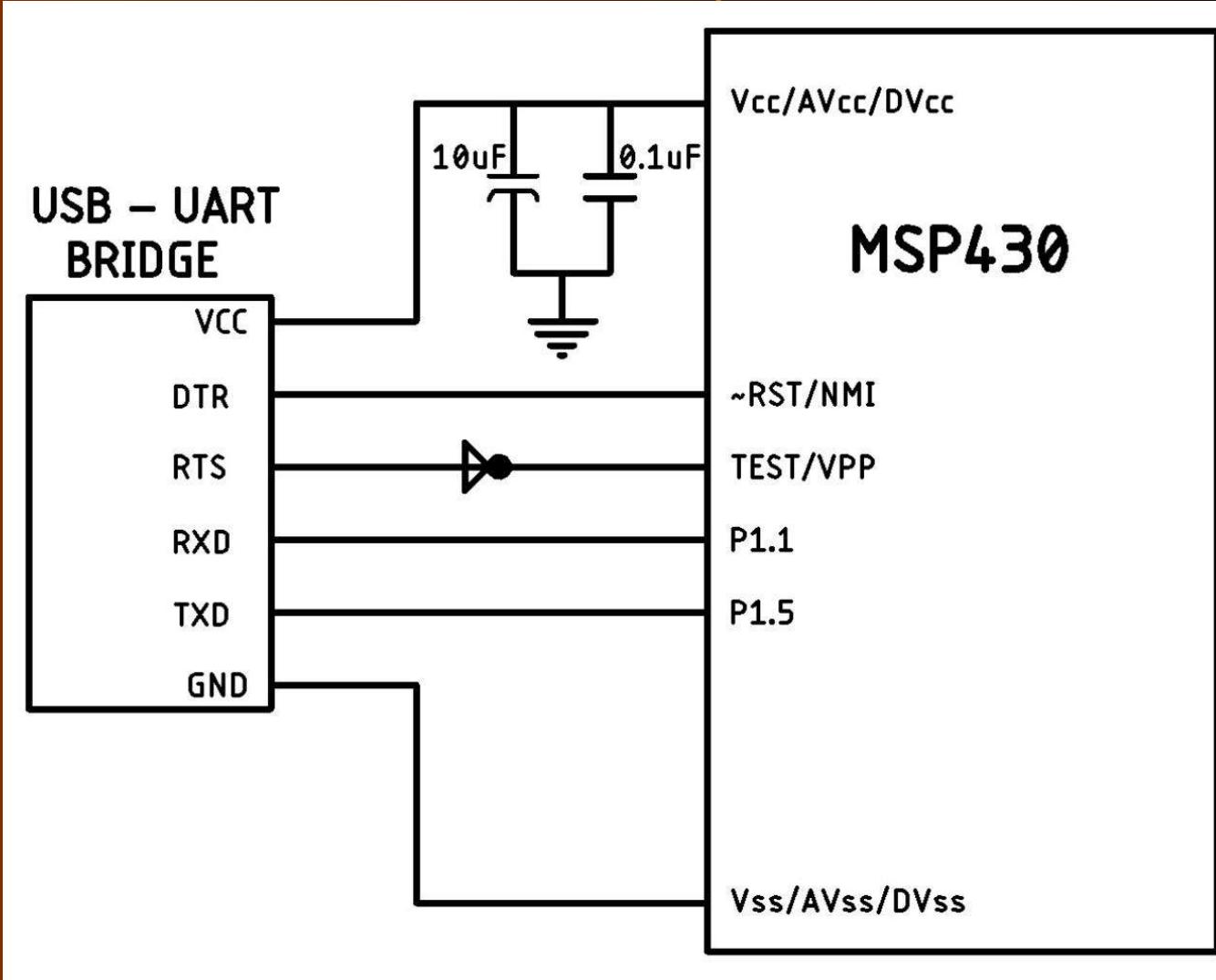
JTAG Debugger



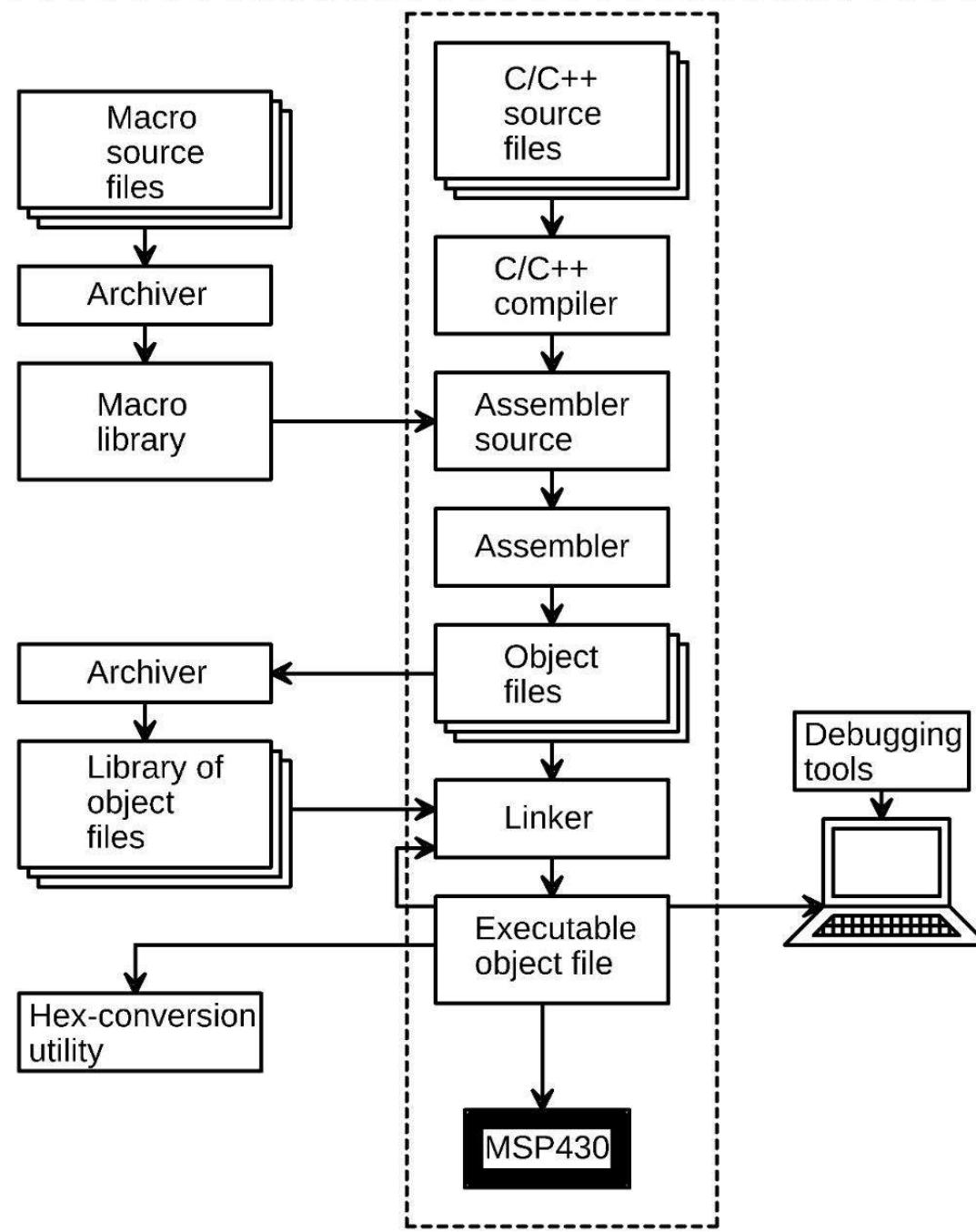
BSL

- The MSP430™ bootloader (BSL) (formerly known as the bootstrap loader)
- Allows users to communicate with embedded memory in the MSP430 microcontroller (MCU) during the prototyping phase, final production, and in service
- Both the programmable memory (flash memory) and the data memory (RAM) can be modified as required.

USB/UART Bootloaders



PROGRAM FLOW



Writing Programs for the MSP430

- Assembly Language Programming
- Register-level Access using TI/GCC Compilers
- Using APIs and Third Party Libraries

Usage in this Course

- Code Composer Studio as Software Development Tool
- UART Bootloader as program download mechanism
- Register level and third party API/Libraries programming technique.

Why do we prefer high level language for programming microcontrollers?



Thank you!

Introduction to Embedded System Design

Physical Interfacing-1

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

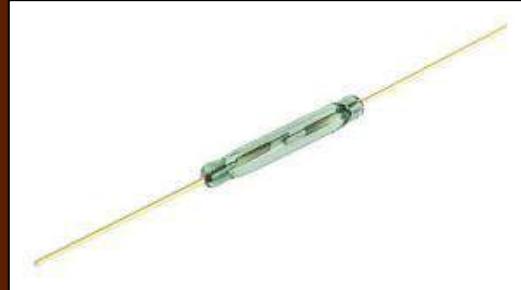
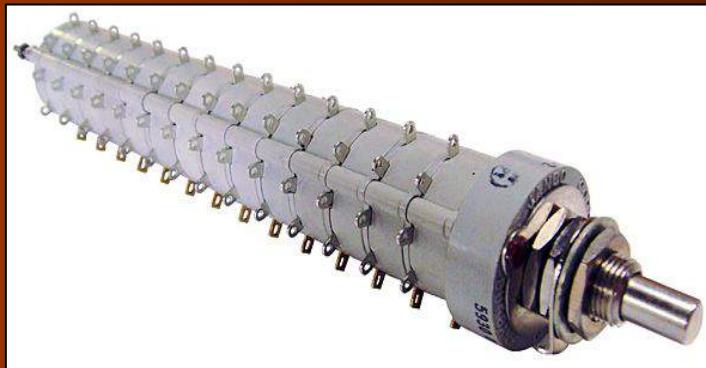
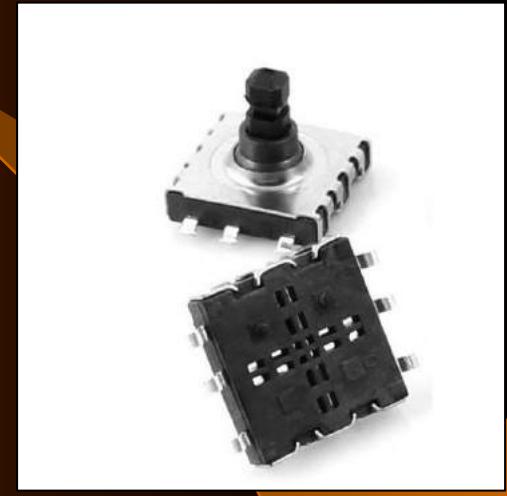
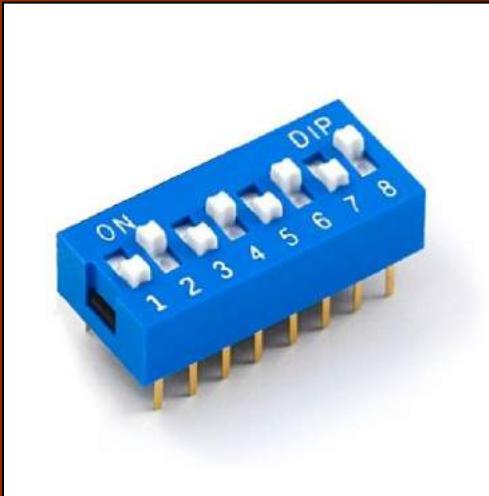
Electrical Engineering Department

Indian Institute of Technology,
Jammu

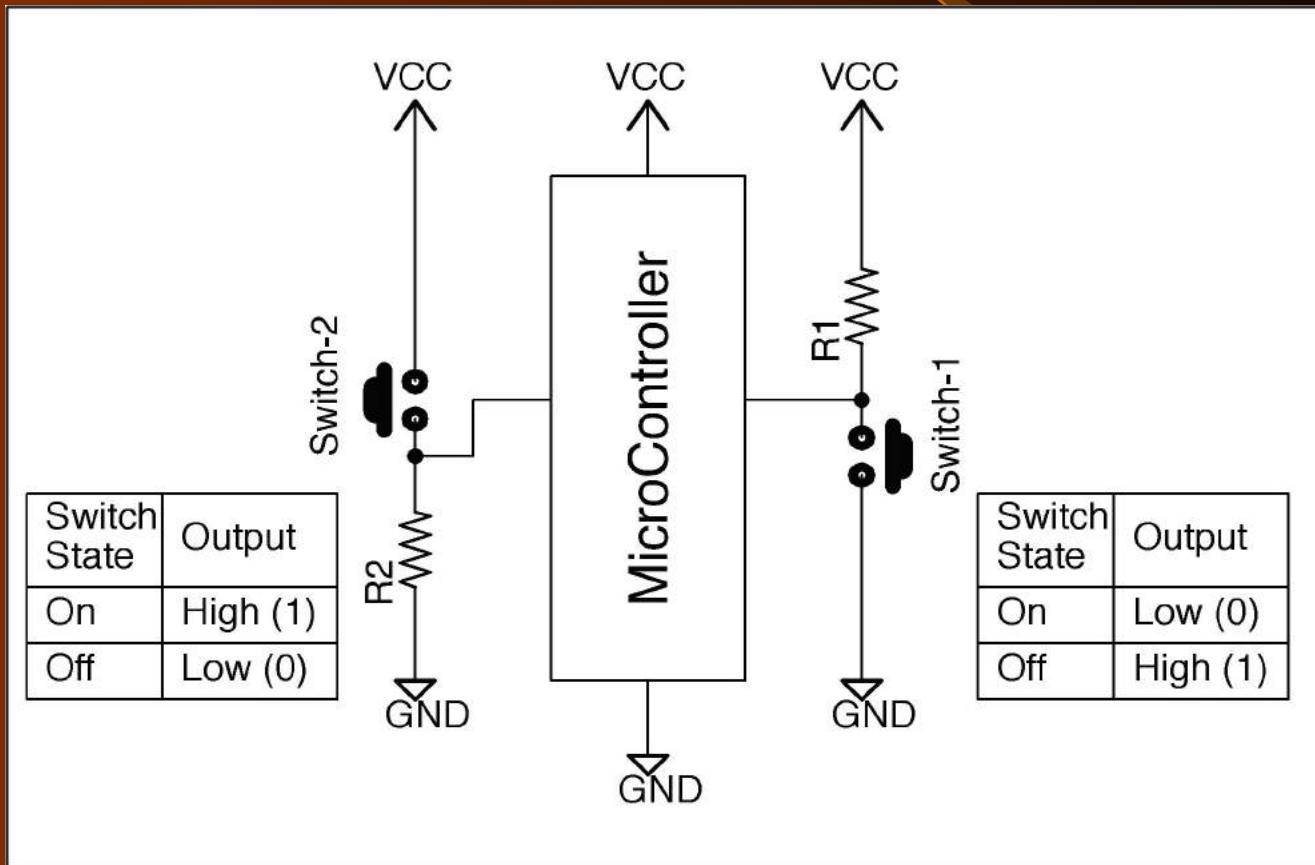
Interfacing to the Physical World: Input Devices

- Input Devices (Human Inputs)
Push Button, Toggle switch, MPMT, Keypad (Matrix of push buttons), DIP Switch, Capacitive/Resistive Touch, JoyStick, Rotary Encoder (Absolute or Incremental)
- Environment Inputs
Sound, Light, Temperature, Humidity, pH, air flow etc.

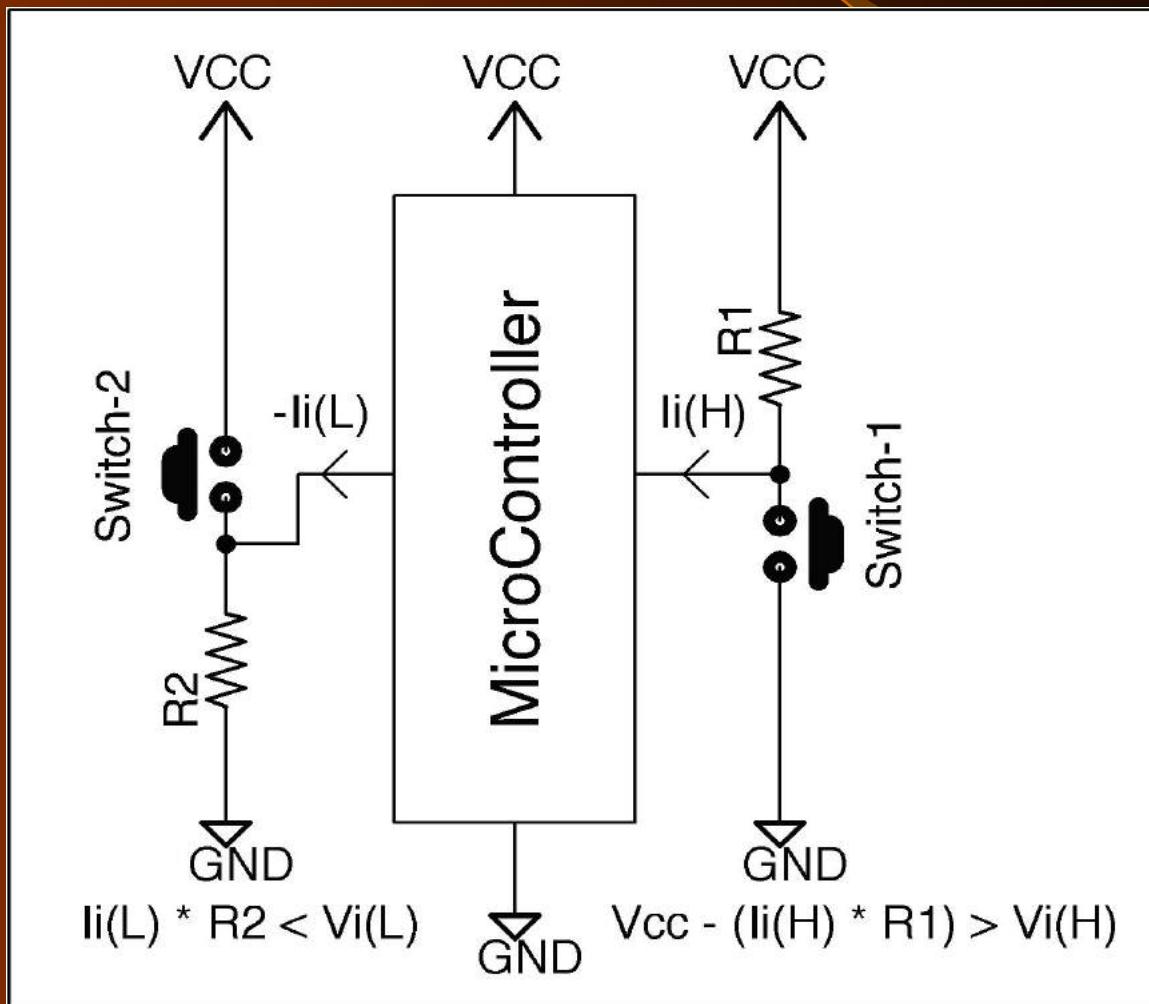
Interfacing to the Physical World: Switches



Connecting Switches



Pull-up and Pull-Down Resistor Values



Schmitt-Trigger Inputs, Ports Px

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER	TEST CONDITIONS	V _{CC}	MIN	TYP	MAX	UNIT
V _{IT+} Positive-going input threshold voltage		3 V	0.45 V _{CC}	0.75 V _{CC}	2.25	V
			1.35			
V _{IT-} Negative-going input threshold voltage		3 V	0.25 V _{CC}	0.55 V _{CC}	1.65	V
			0.75			
V _{hys} Input voltage hysteresis (V _{IT+} – V _{IT-})		3 V	0.3		1	V
R _{Pull} Pullup/pulldown resistor	For pullup: V _{IN} = V _{SS} For pulldown: V _{IN} = V _{CC}	3 V	20	35	50	kΩ
C _I Input capacitance	V _{IN} = V _{SS} or V _{CC}			5		pF

Leakage Current, Ports Px

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER	TEST CONDITIONS	V _{CC}	MIN	MAX	UNIT
I _{lkp(Px,y)} High-impedance leakage current	(1) (2)	3 V		±50	nA

- (1) The leakage current is measured with V_{SS} or V_{CC} applied to the corresponding pin(s), unless otherwise noted.
- (2) The leakage of the digital port pins is measured individually. The port pin is selected for input and the pullup/pulldown resistor is disabled.

Outputs, Ports Px

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER	TEST CONDITIONS	V _{CC}	MIN	TYP	MAX	UNIT
V _{OH} High-level output voltage	I _(OHmax) = -6 mA ⁽¹⁾	3 V		V _{CC} – 0.3		V
V _{OL} Low-level output voltage	I _(OLmax) = 6 mA ⁽¹⁾	3 V		V _{SS} + 0.3		V

- (1) The maximum total current, I_(OHmax) and I_(OLmax), for all outputs combined should not exceed ±48 mA to hold the maximum voltage drop specified.

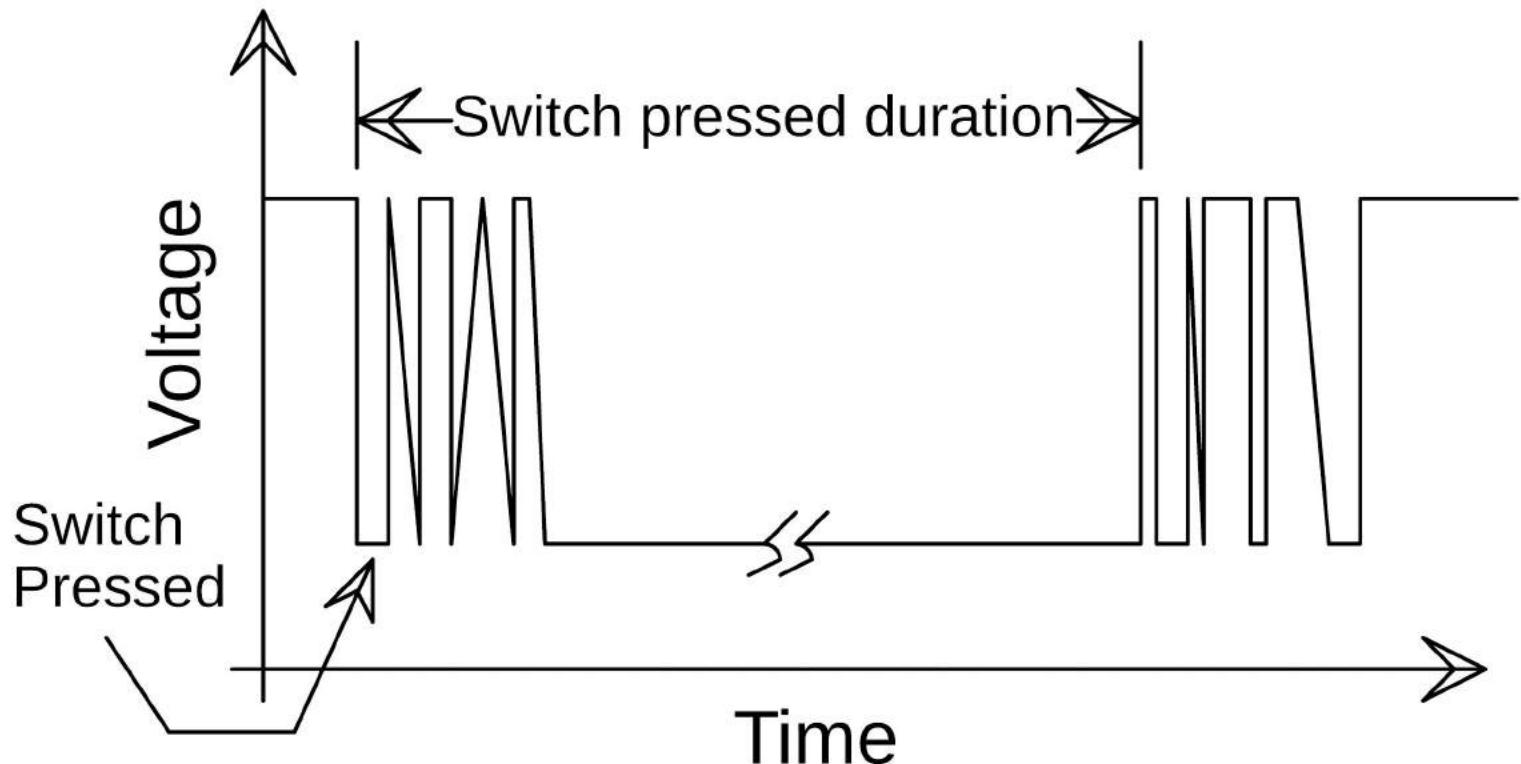
Output Frequency, Ports Px

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

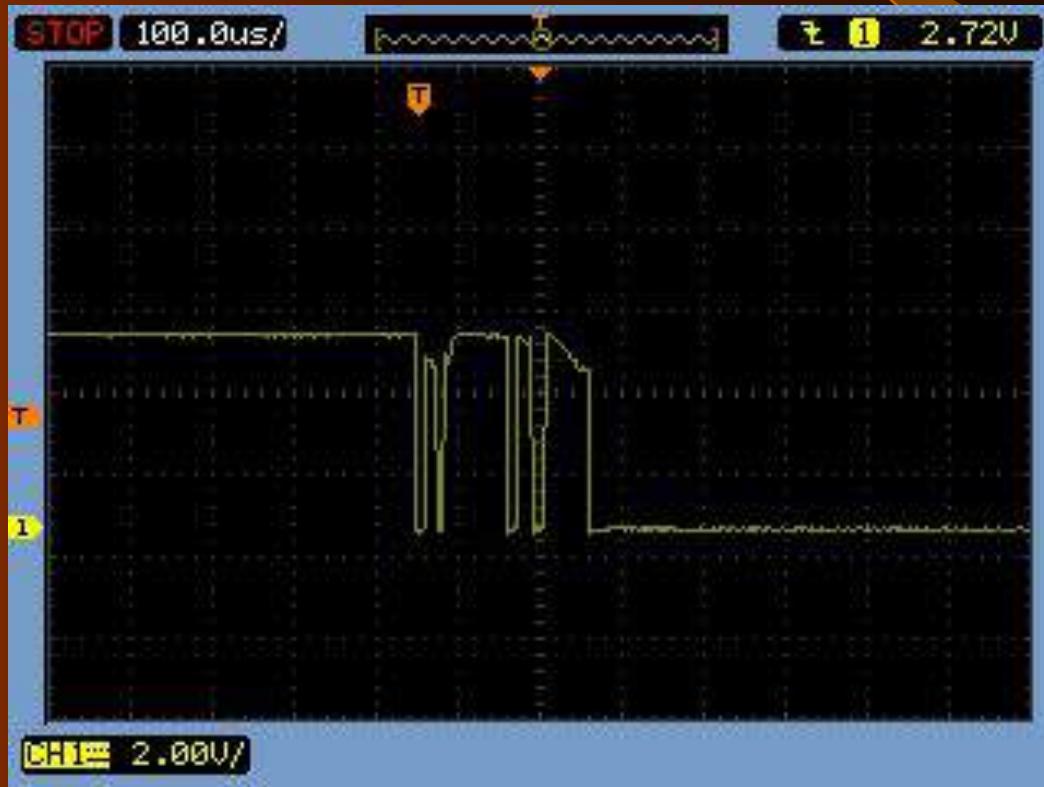
PARAMETER	TEST CONDITIONS	V _{CC}	MIN	TYP	MAX	UNIT
f _{Px,y} Port output frequency (with load)	Px.y, C _L = 20 pF, R _L = 1 kΩ ⁽¹⁾ (2)	3 V		12		MHz
f _{Port_CLK} Clock output frequency	Px.y, C _L = 20 pF ⁽²⁾	3 V		16		MHz

- (1) A resistive divider with two 0.5-kΩ resistors between V_{CC} and V_{SS} is used as load. The output is connected to the center tap of the divider.
- (2) The output voltage reaches at least 10% and 90% V_{CC} at the specified toggle frequency.

Switch Bounce (With Pull-up Resistor)



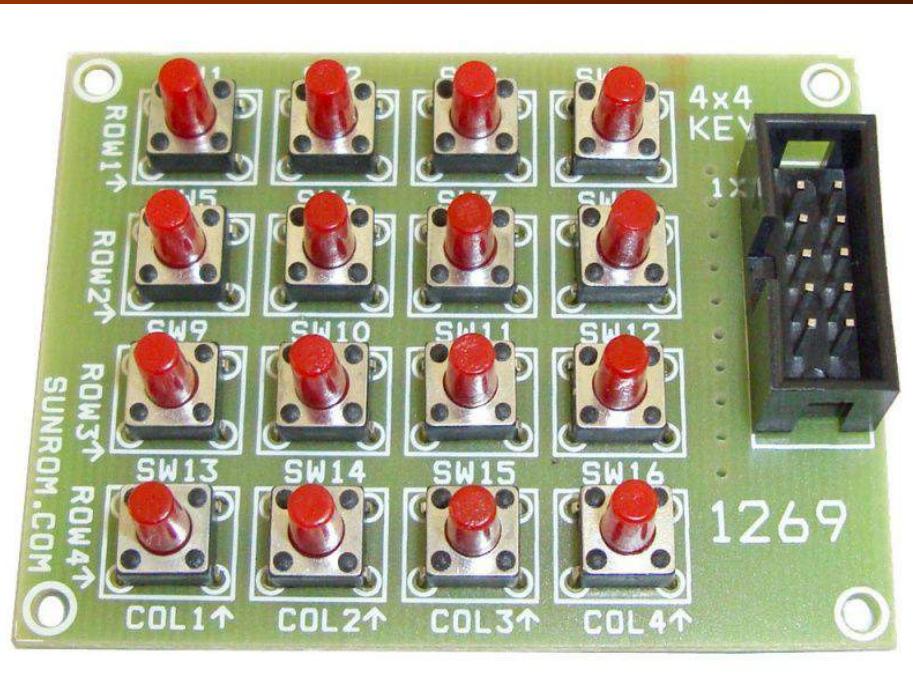
Switch Bounce DSO Screen Capture



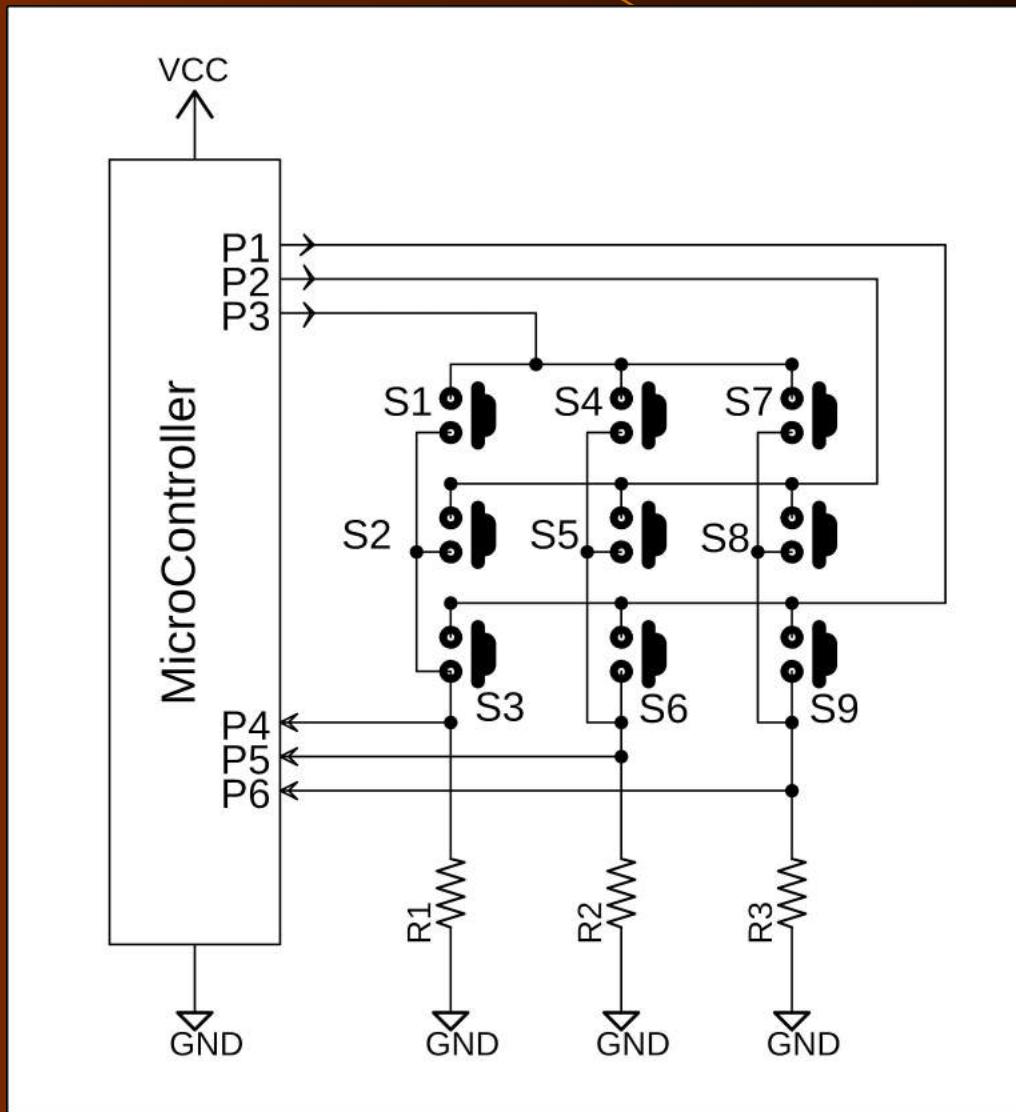
Switch Debounce

- Hardware option (not preferred)
- Software option

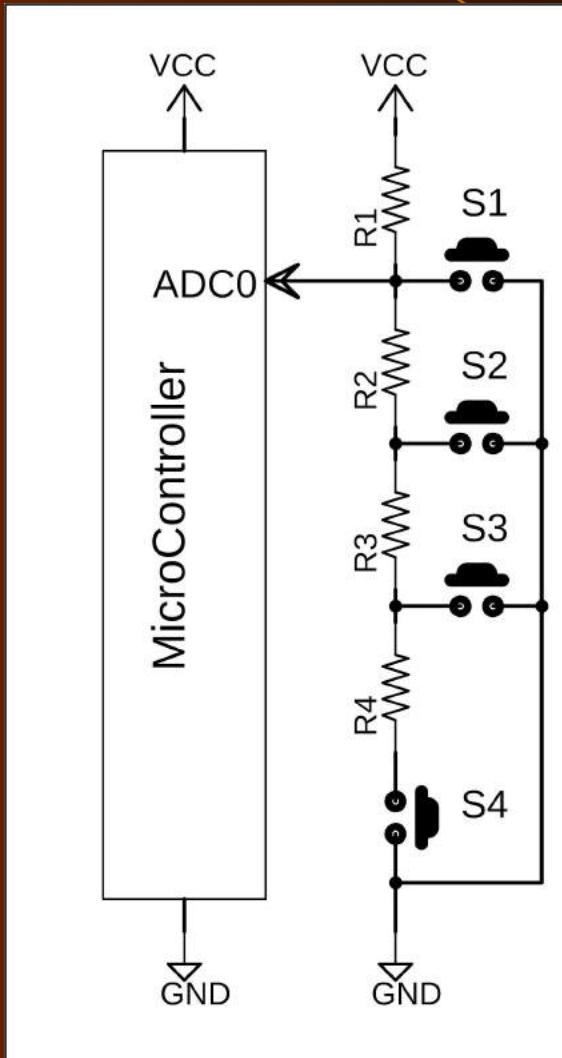
Interfacing to the Physical World: More Inputs



Matrix of Keys

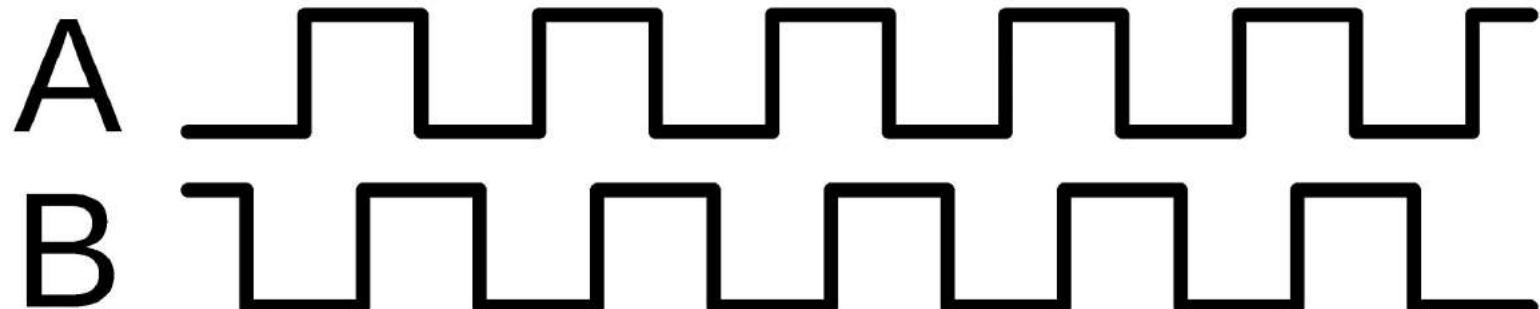


Using an ADC To Read Multiple Switches



Rotary Encoder

Rotary Encoder



Interfacing to the Physical World: Output Devices

- Output Devices (Human Outputs)

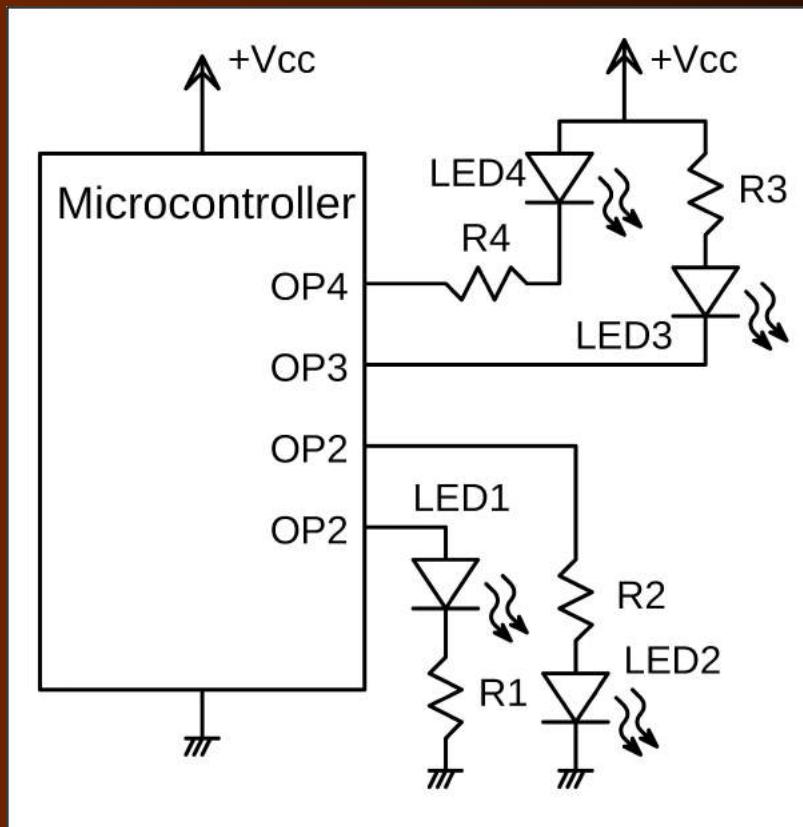
LED, RGB LED, Addressable RGB LED, Seven Segment Display, Dot-Matrix Display, LCD, Sound output

- Other Outputs

Relay, Motor, Heater, Peltier Module, DC Motor, Stepper Motor.

Controlling LEDs

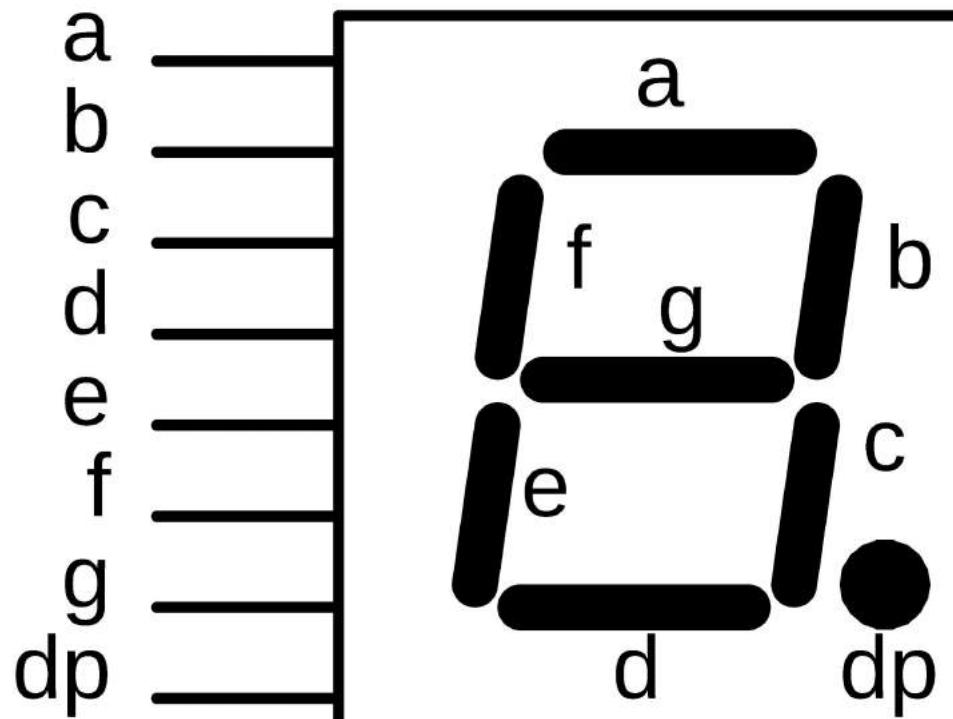
- LED1 and LED2: High Side Control
- LED3 and LED4: Low Side Control



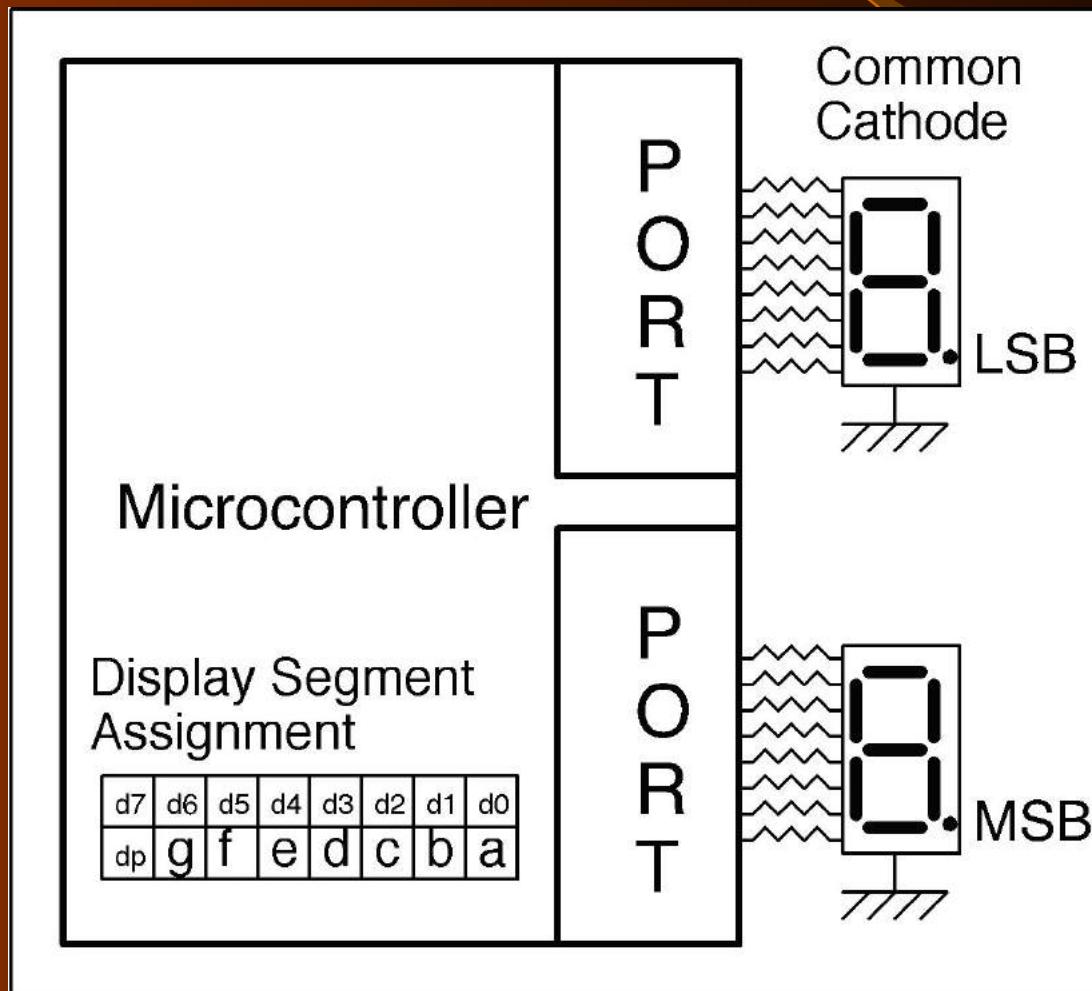
Controlling LEDs

- **What is the voltage drop across an LED?**
- **What is the value of the resistor?**
- **How to calculate the value**

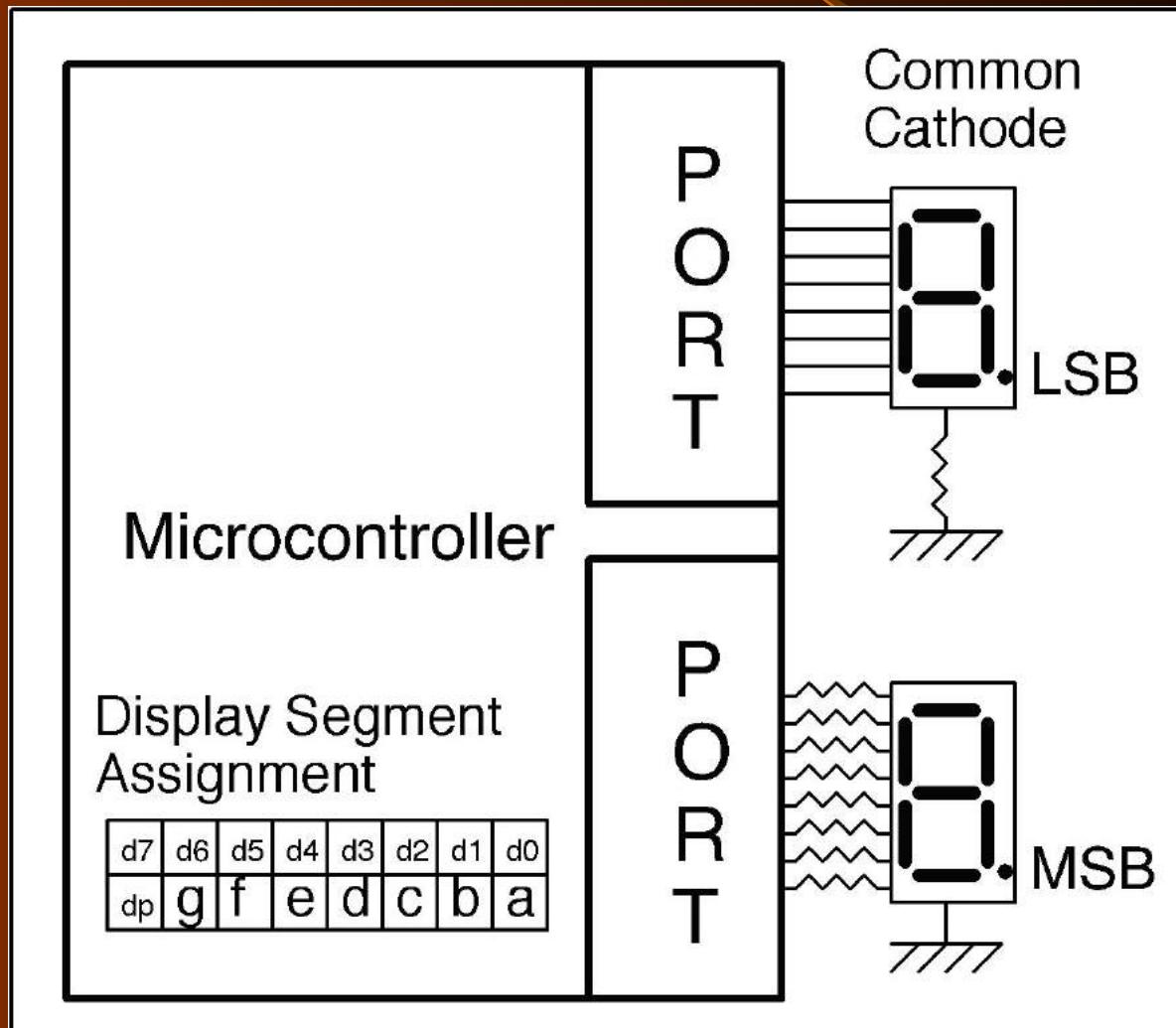
Driving Seven Segment Displays



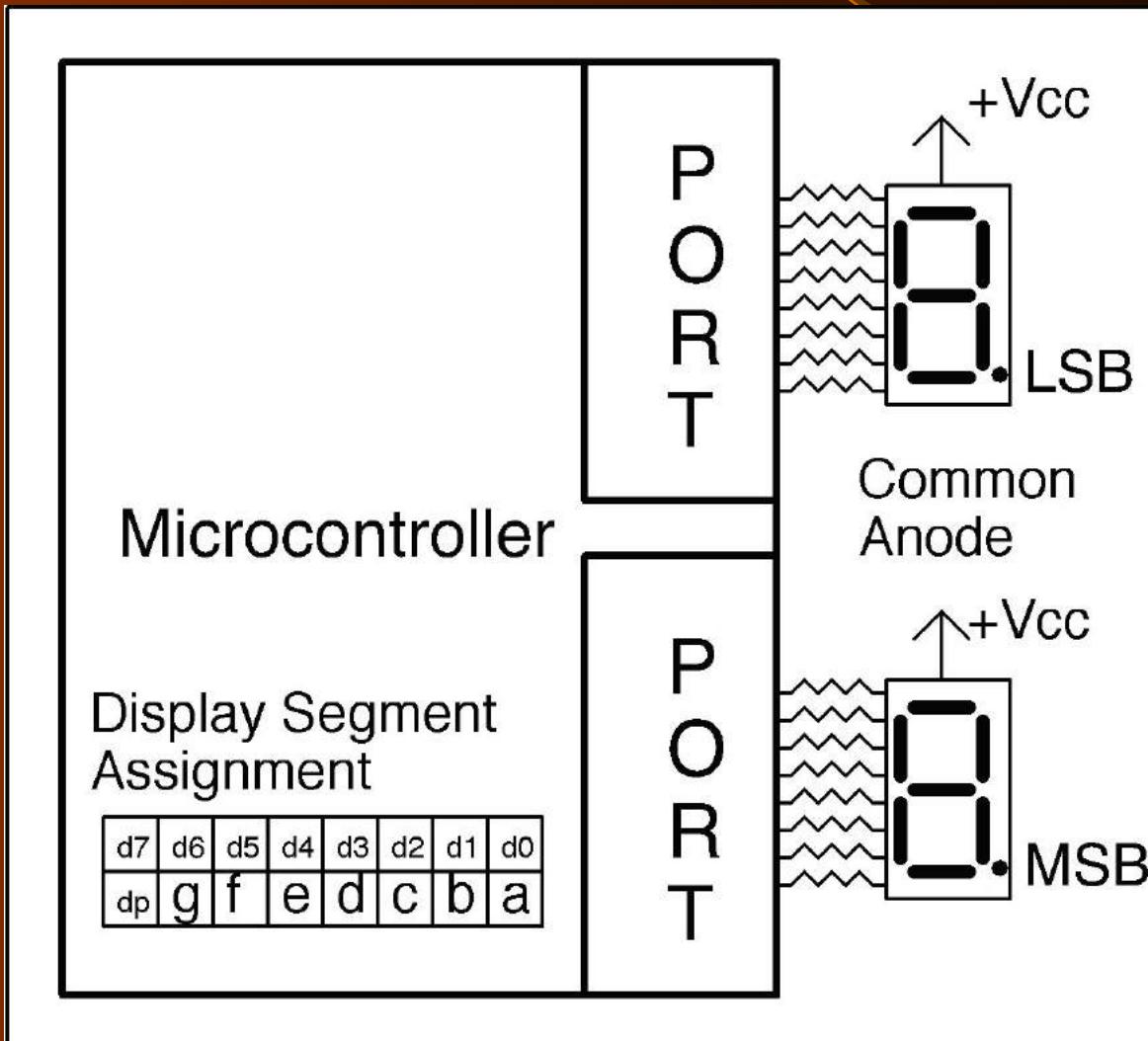
Driving Seven Segment Displays



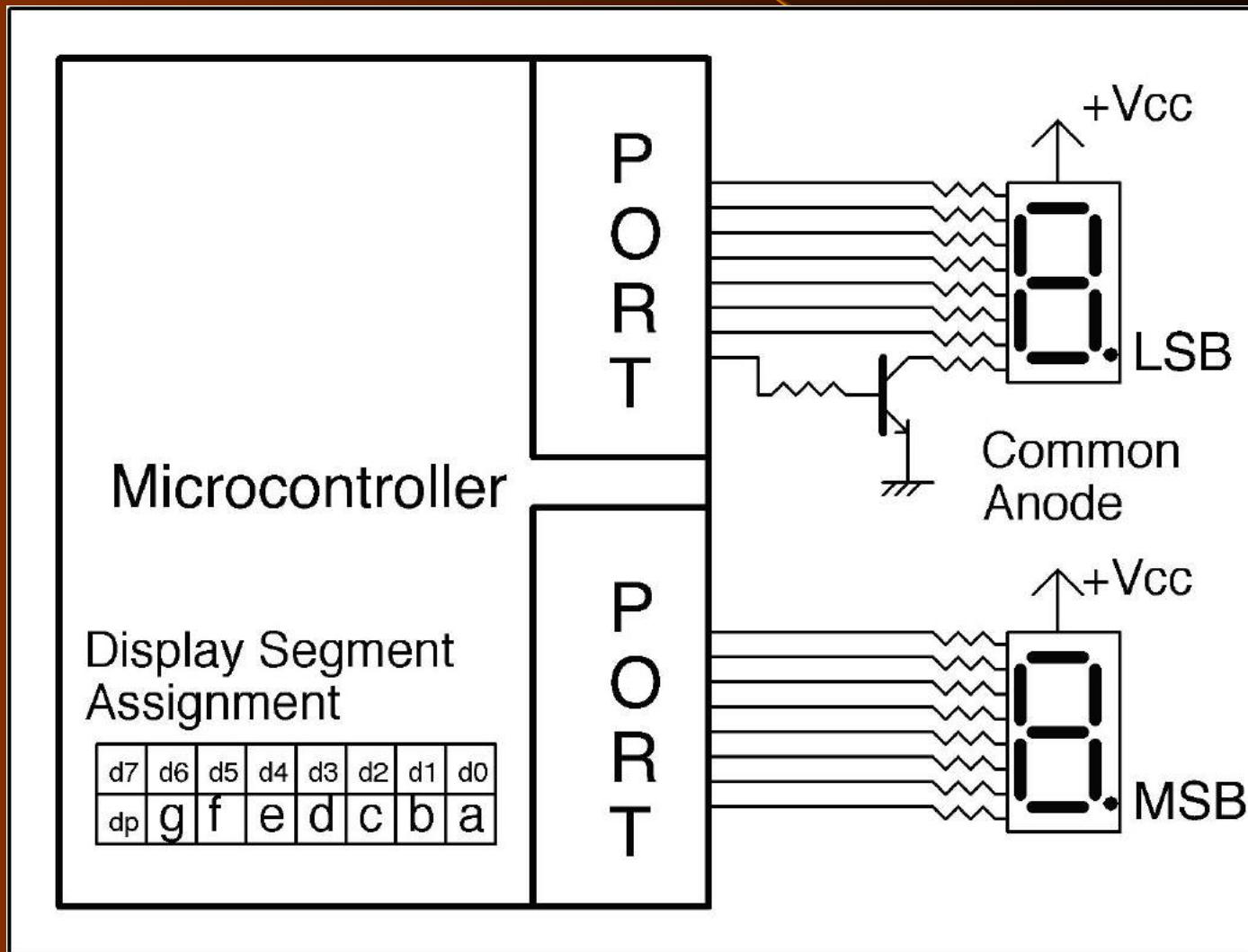
Driving Seven Segment Displays



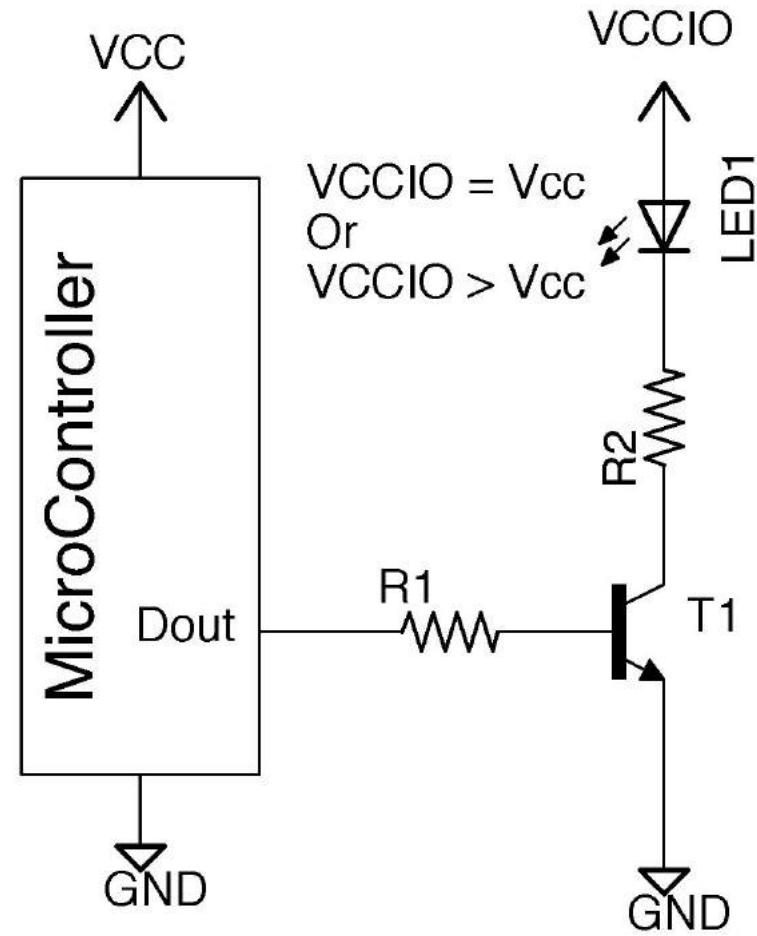
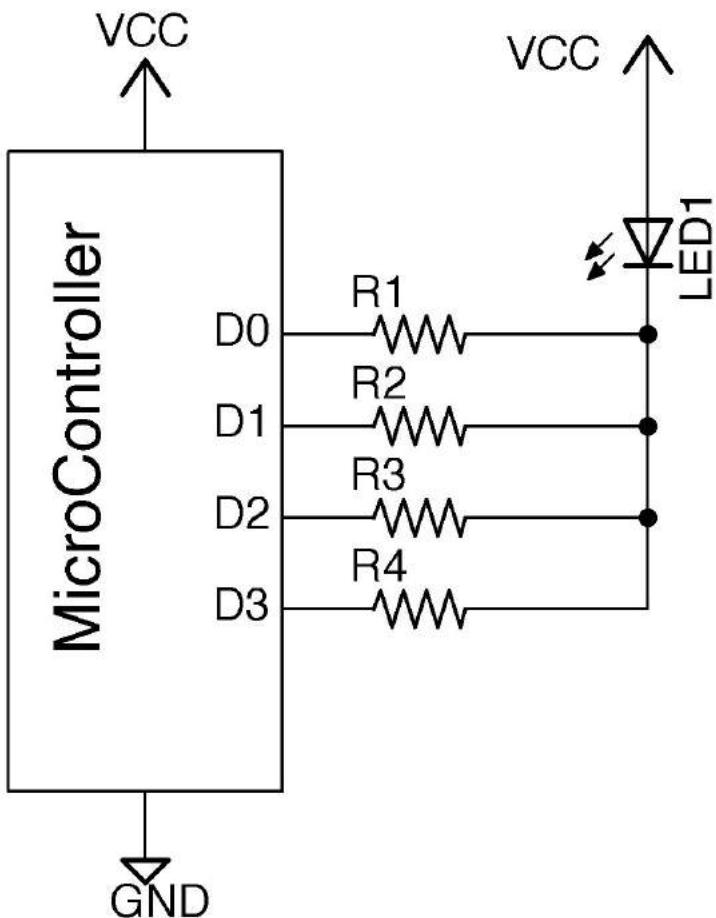
Driving Seven Segment Displays



Driving Seven Segment Displays



Driving LEDs (Or Other loads): Low Side Driver



Introduction to Embedded System Design

Physical Interfacing- 2

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

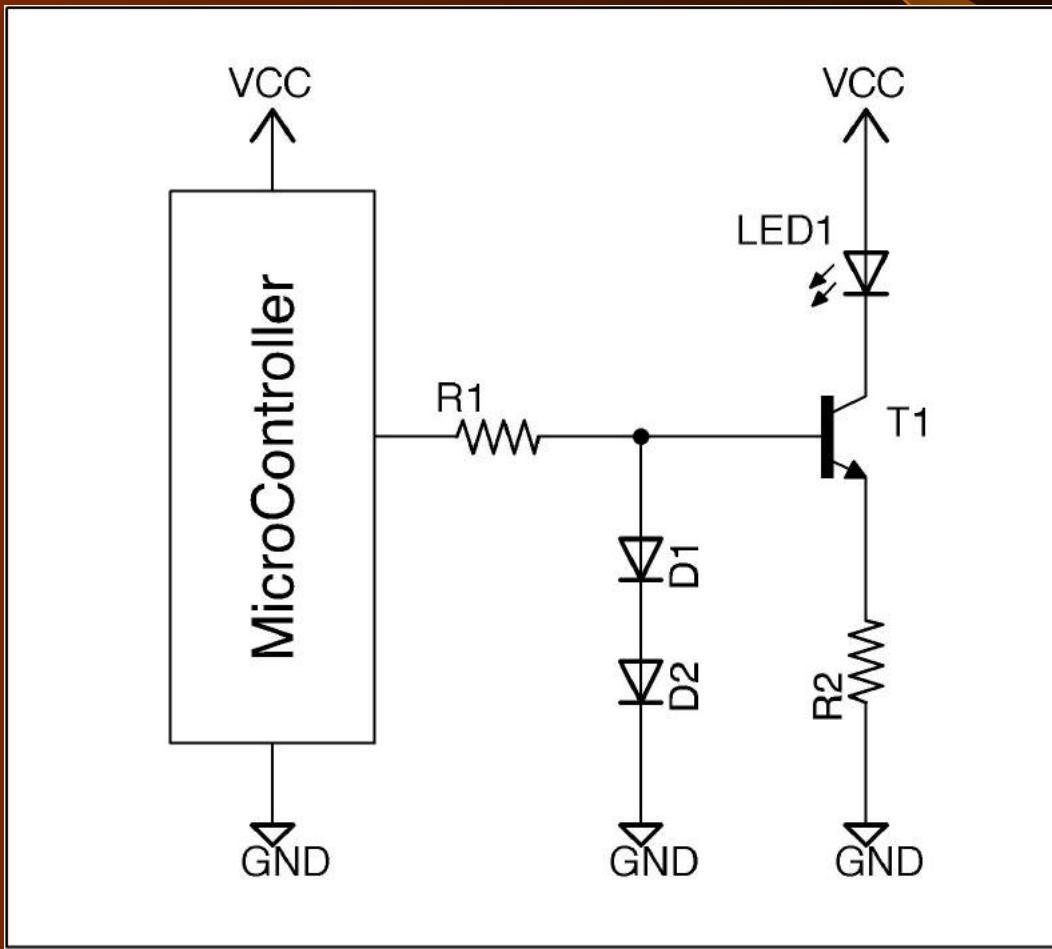
Badri Subudhi

Assistant Professor

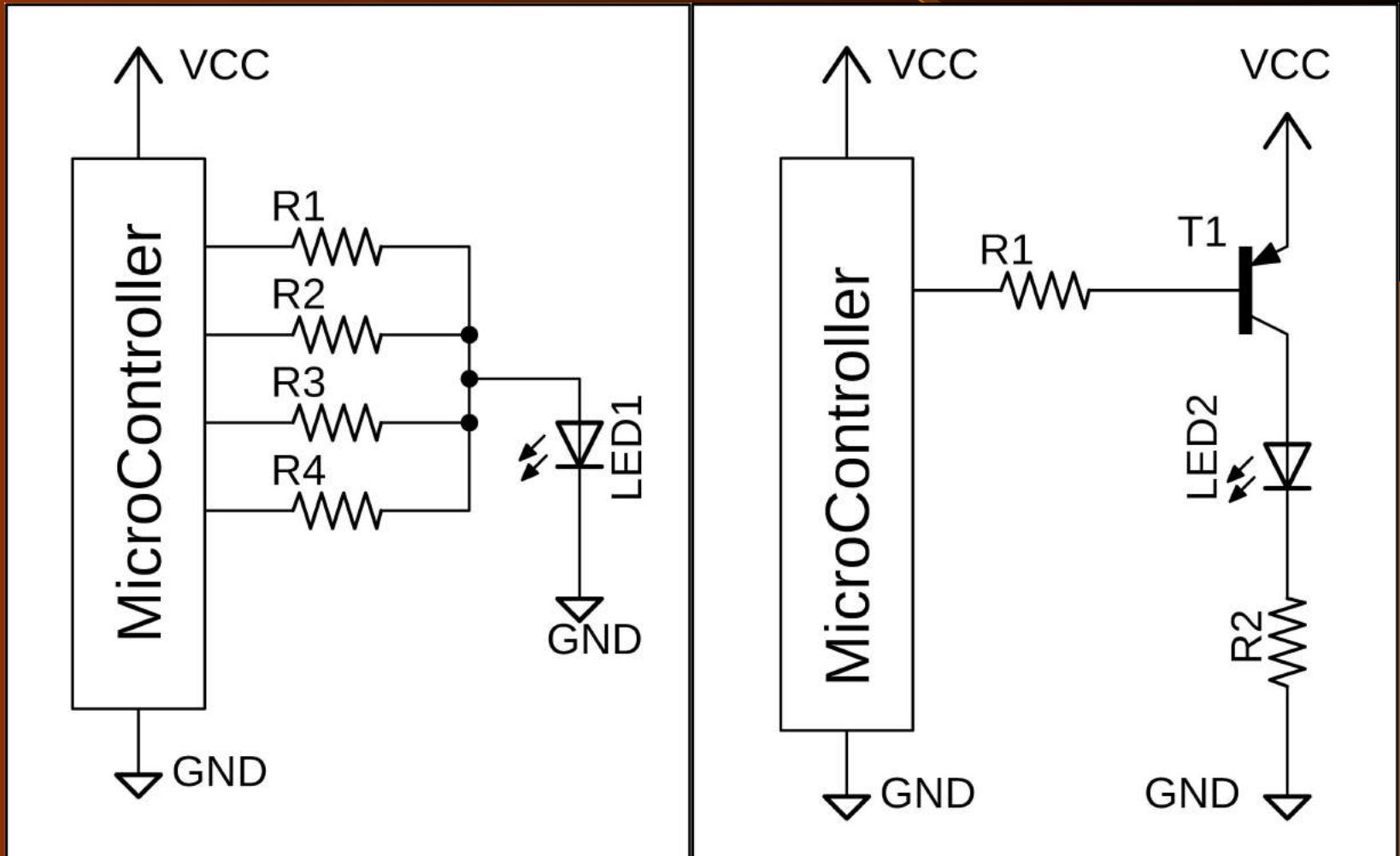
Electrical Engineering Department

Indian Institute of Technology,
Jammu

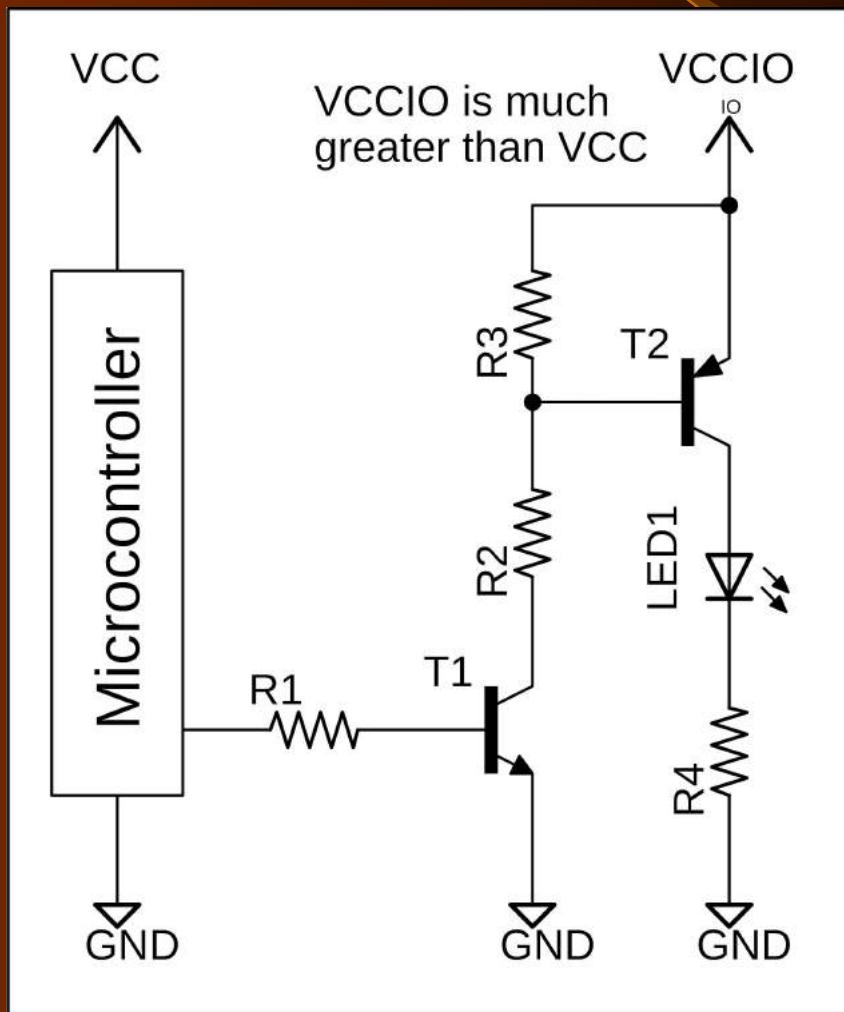
Driving LEDs (Or Other loads): Constant Current Low Side Driver



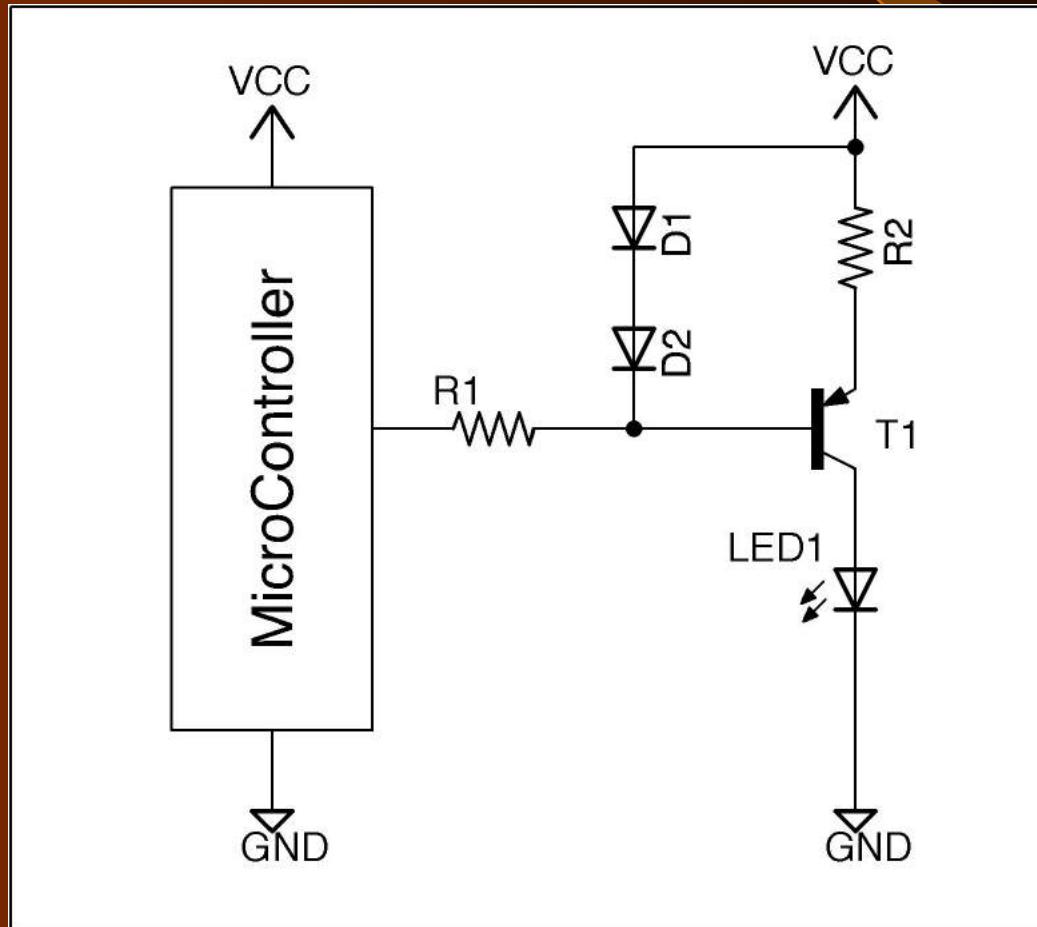
Driving LEDs (Or Other loads): High Side Driver



Driving LEDs (Or Other loads): High Side Driver



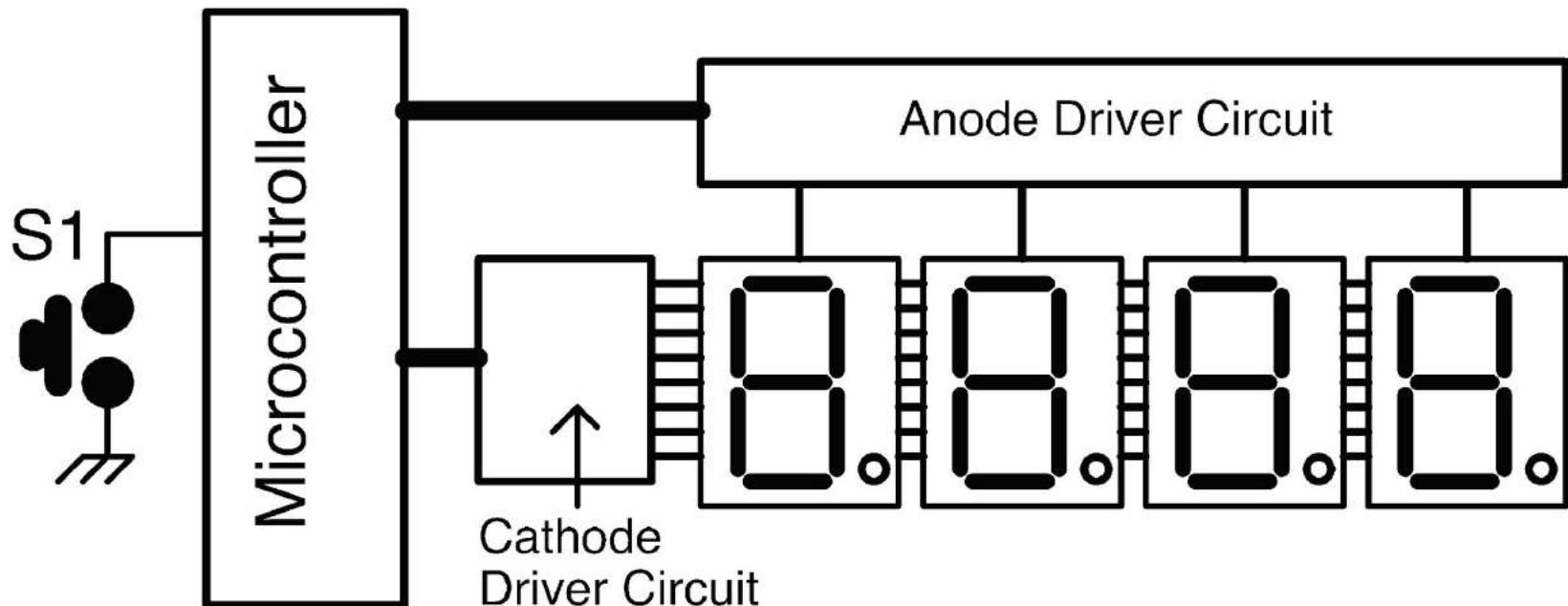
Driving LEDs (Or Other loads): Constant Current High Side Driver



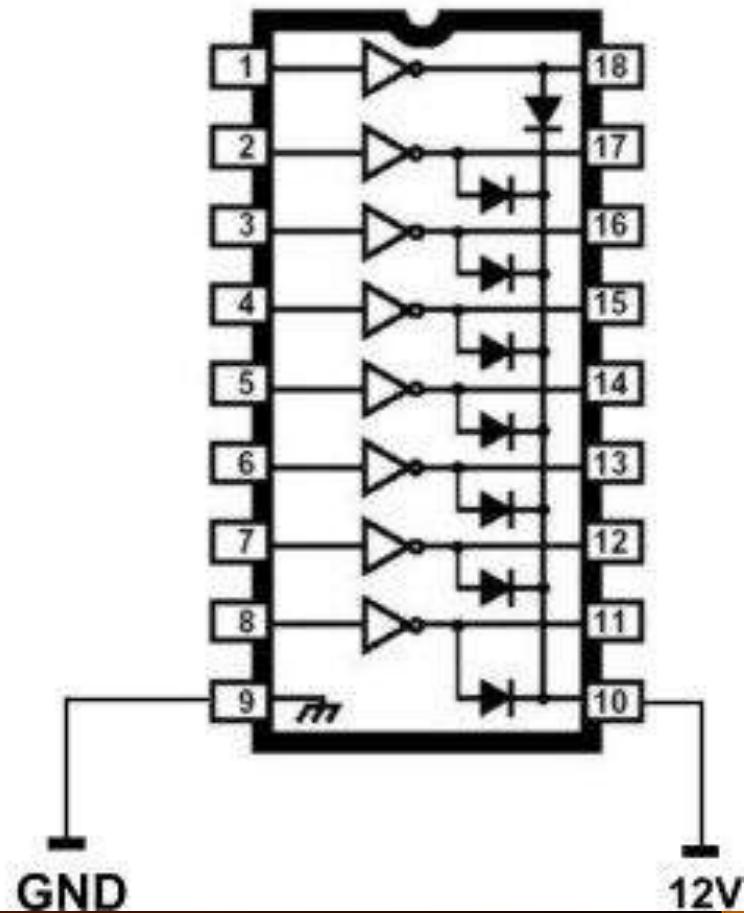
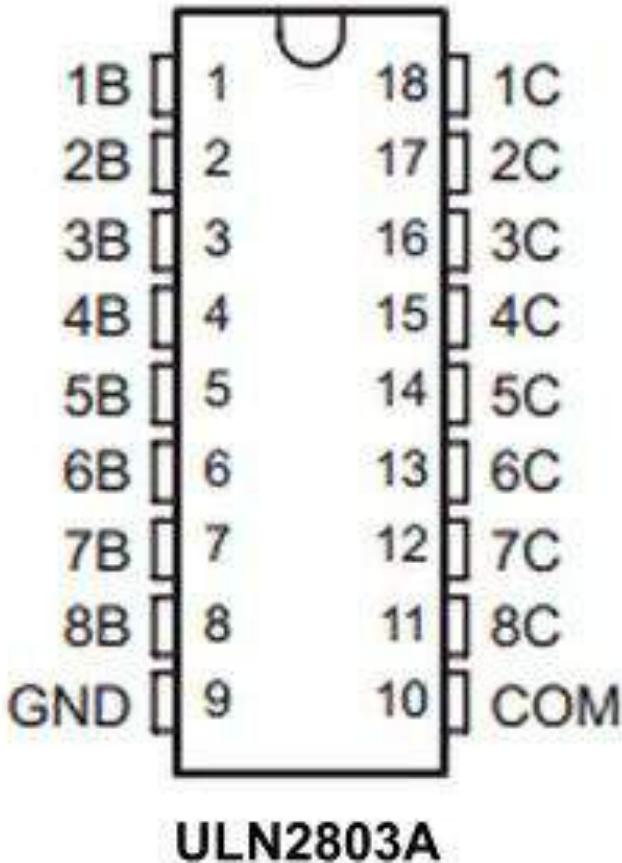
Controlling LEDs

- N pins $\rightarrow N$ LEDs?
- N pins $\rightarrow (N^2)/4$ LEDs?
(Multiplexing)
- N pins $\rightarrow N*(N-1)$ LEDs?
(Charlieplexing)

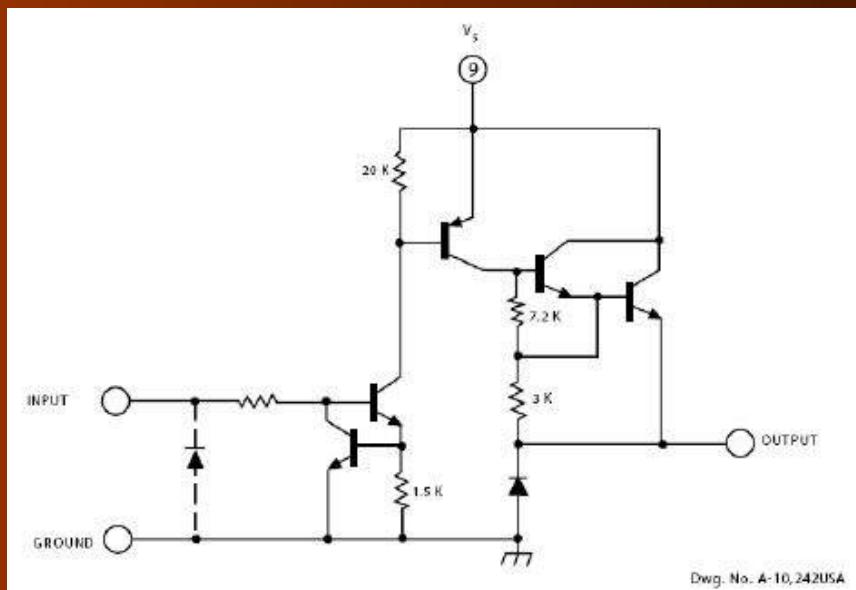
Driving Seven Segment Displays



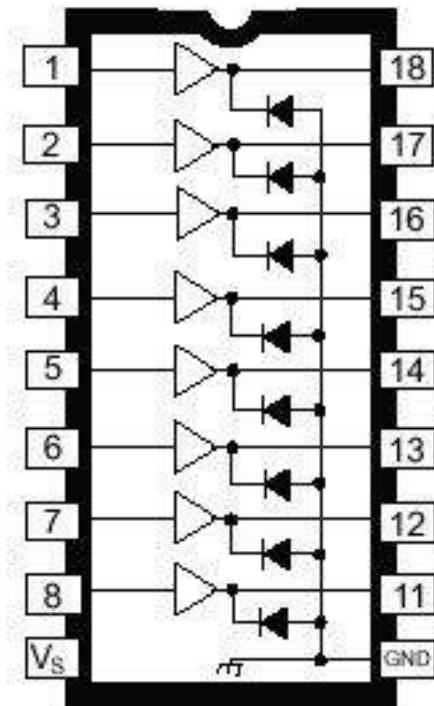
Low Side Driver



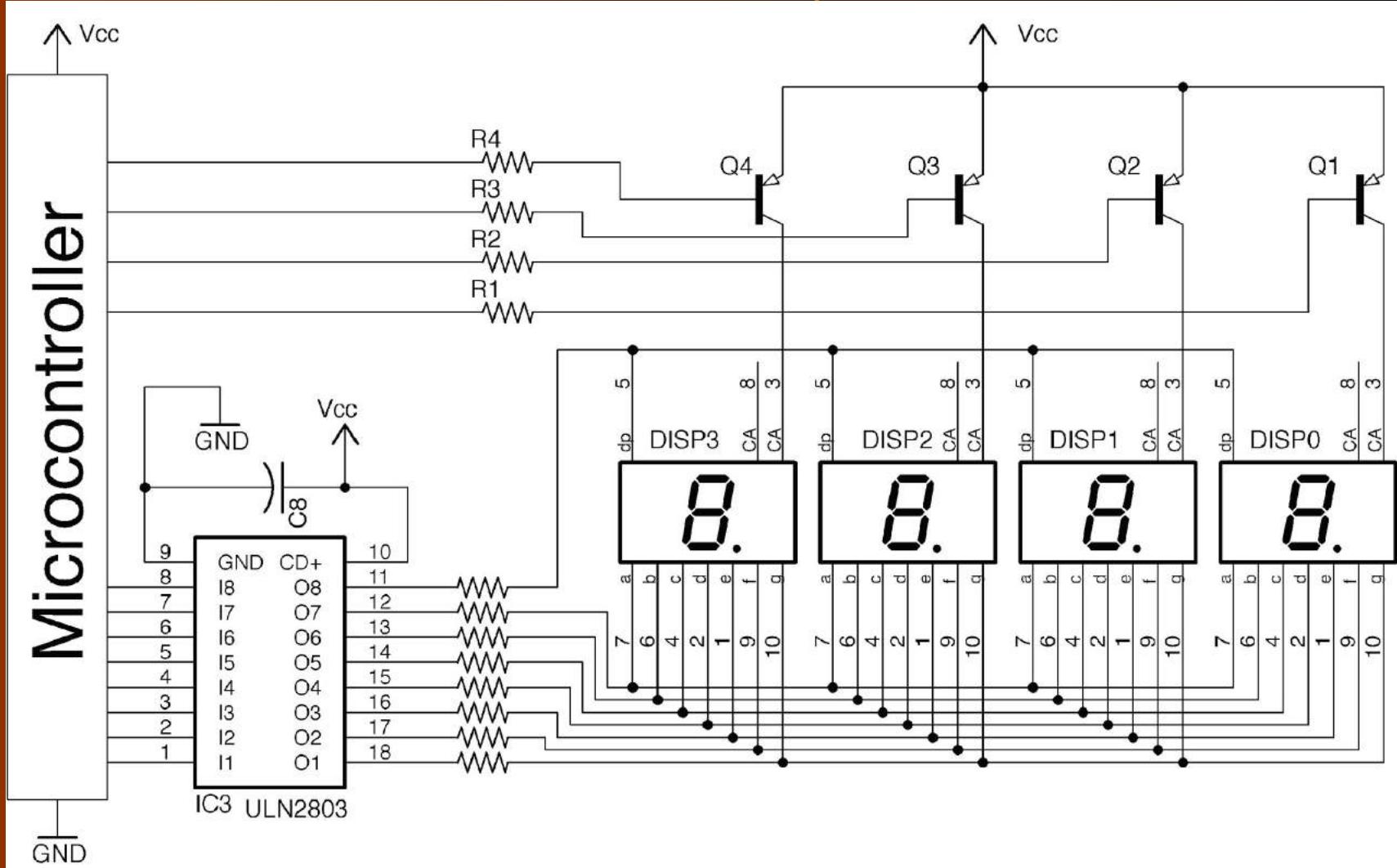
High Side Driver



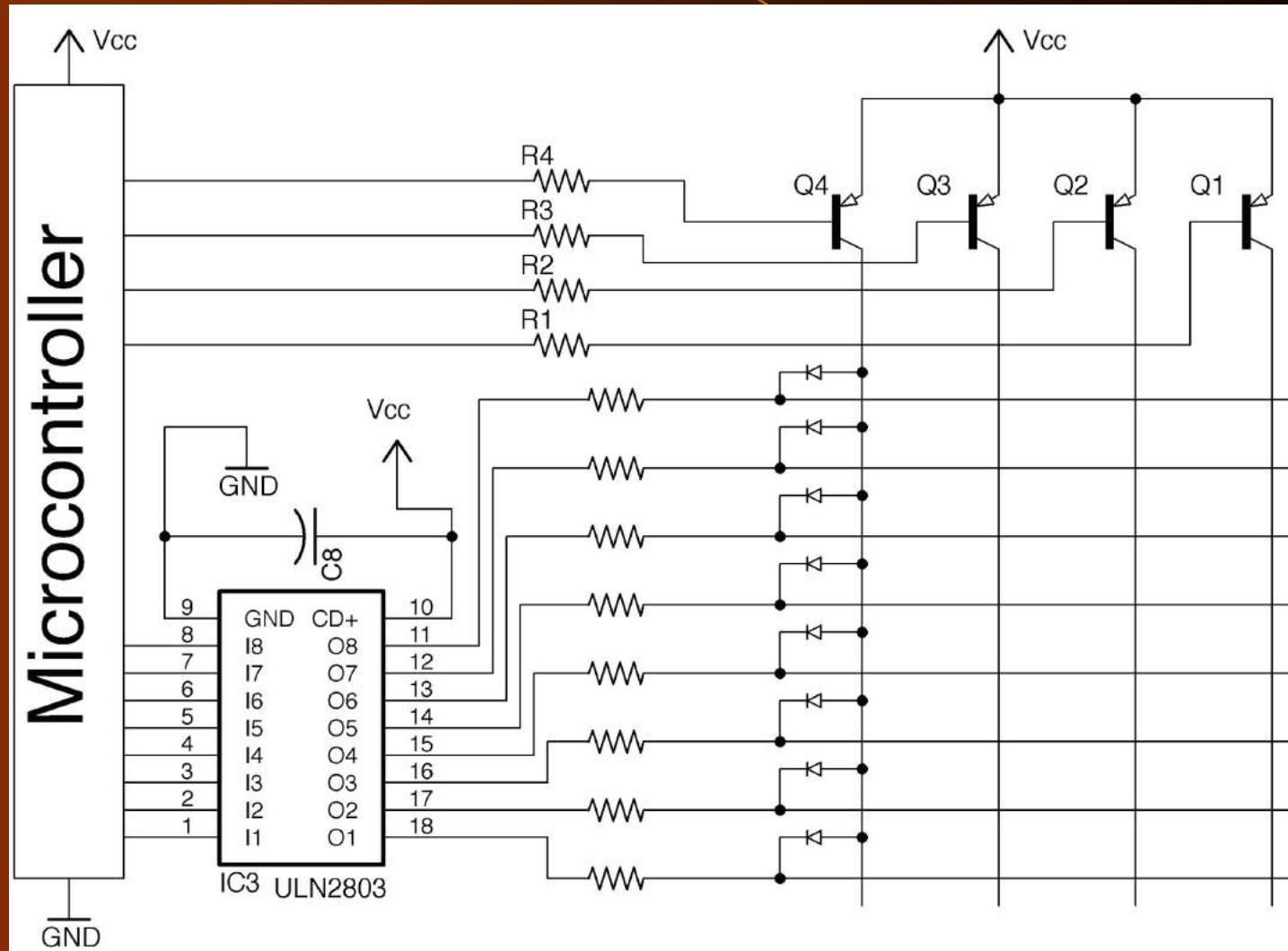
UDN2981A thru UDN2984A



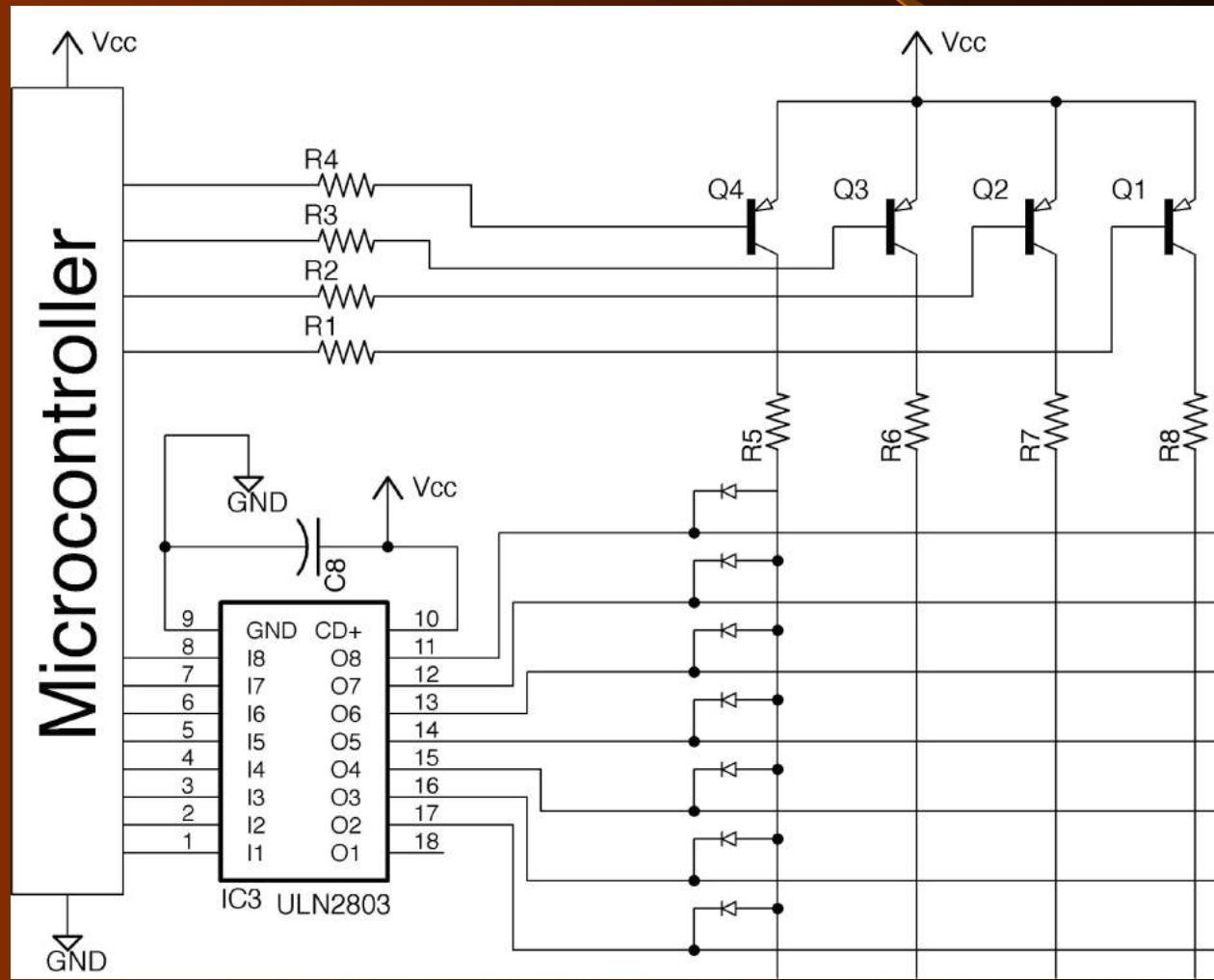
Both Side Switching: Driving Seven Segment Displays



Both Side Switching: Driving Dot Matrix Displays

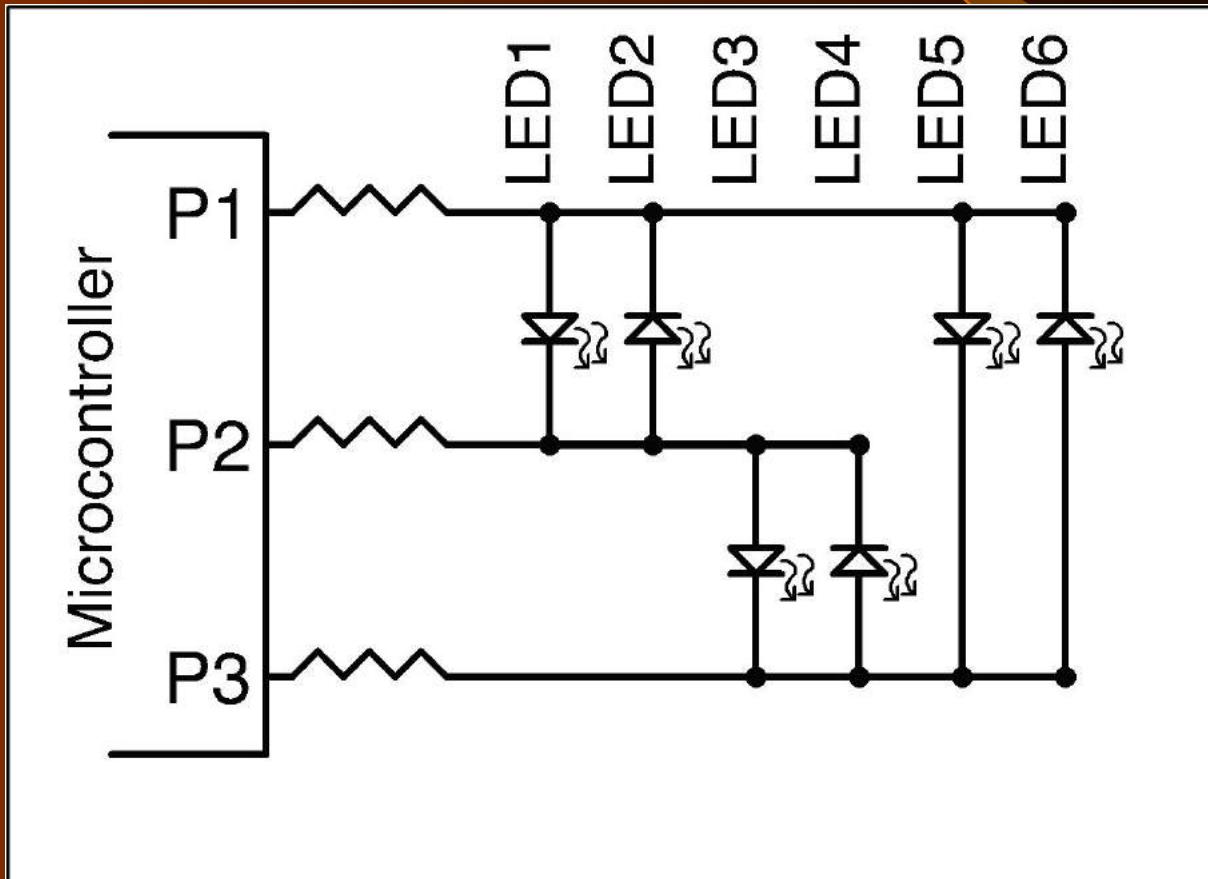


Alternate Method of Both Side Switching

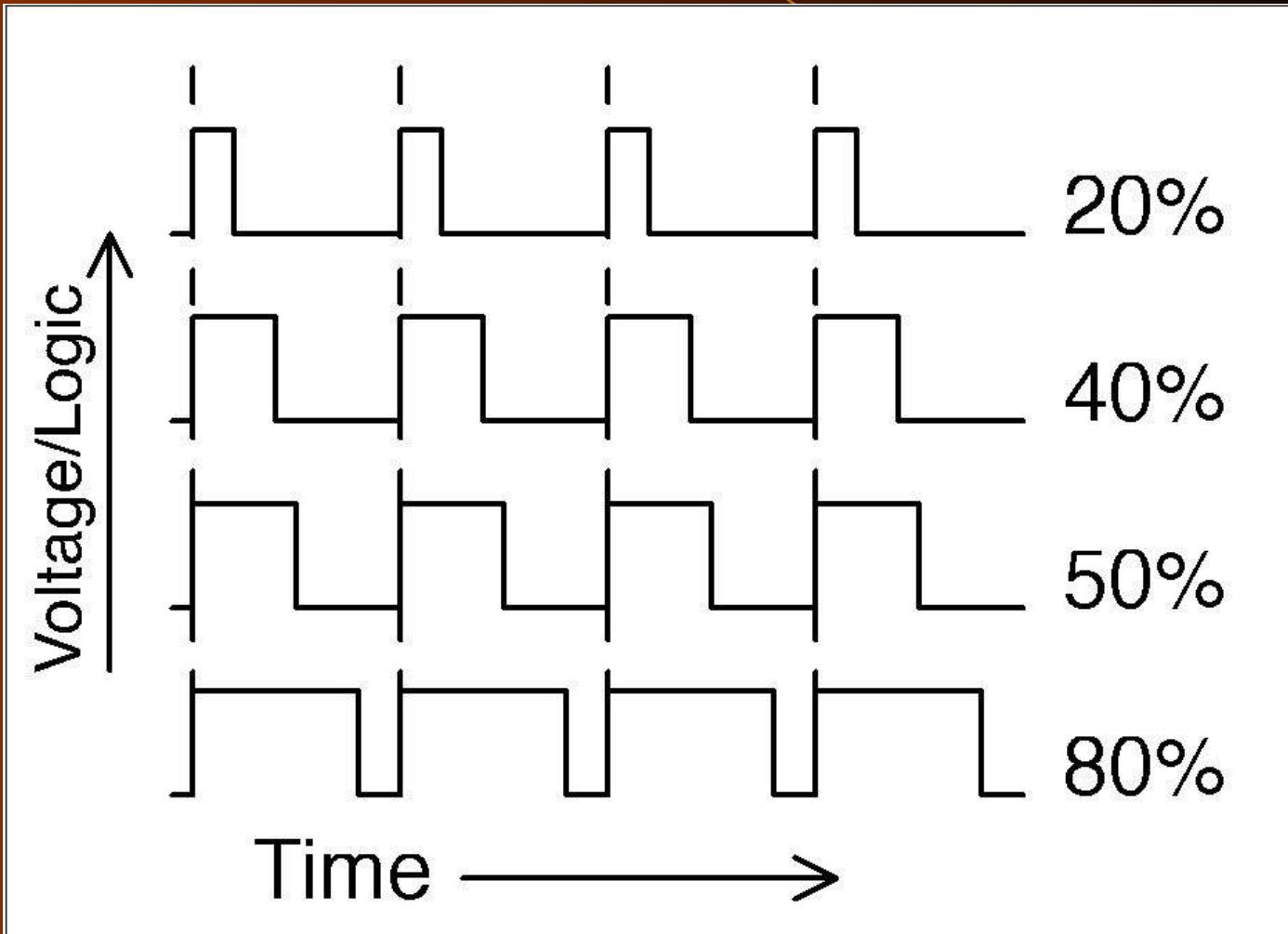


Charlieplexing

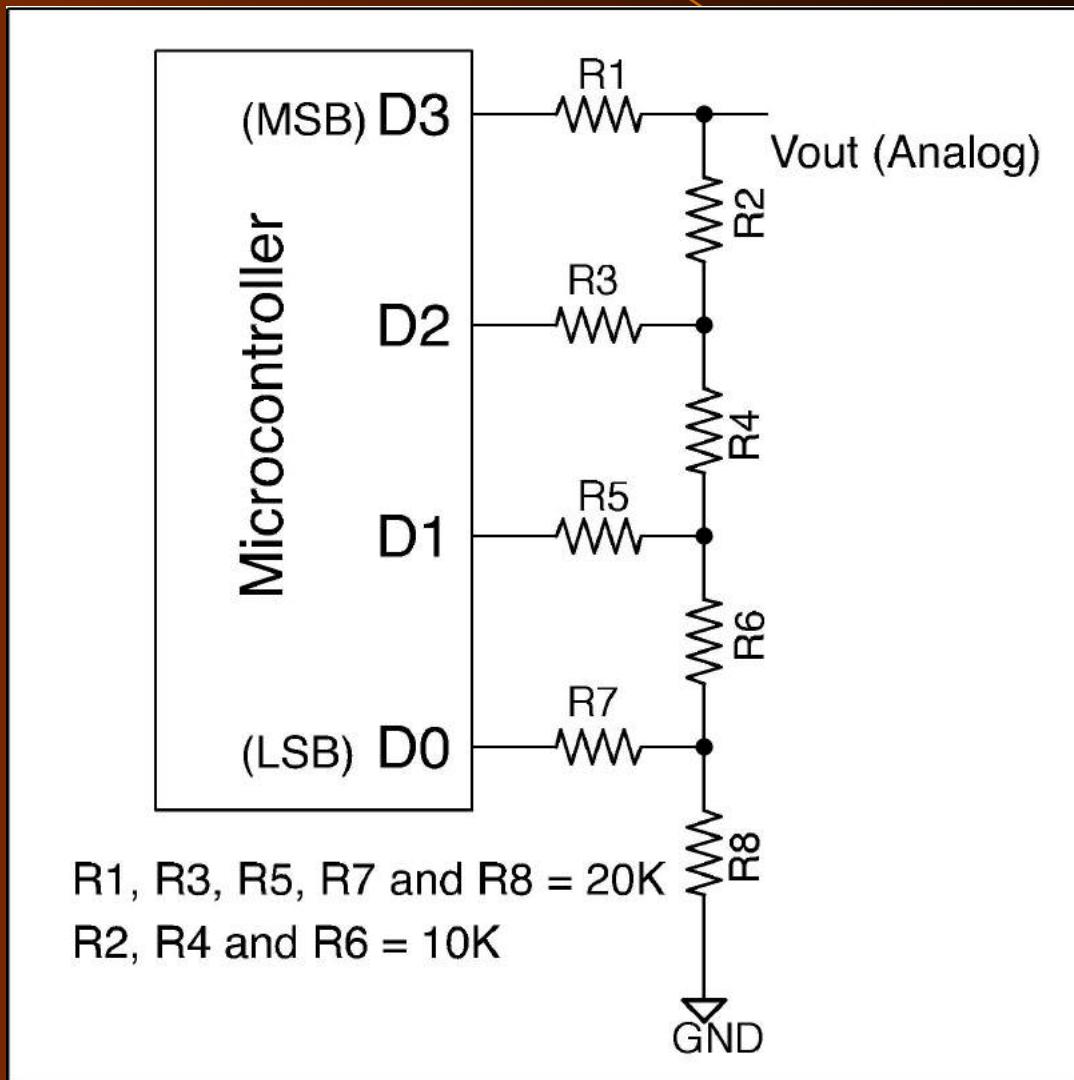
N pins $\rightarrow N^*(N-1)$ LEDs



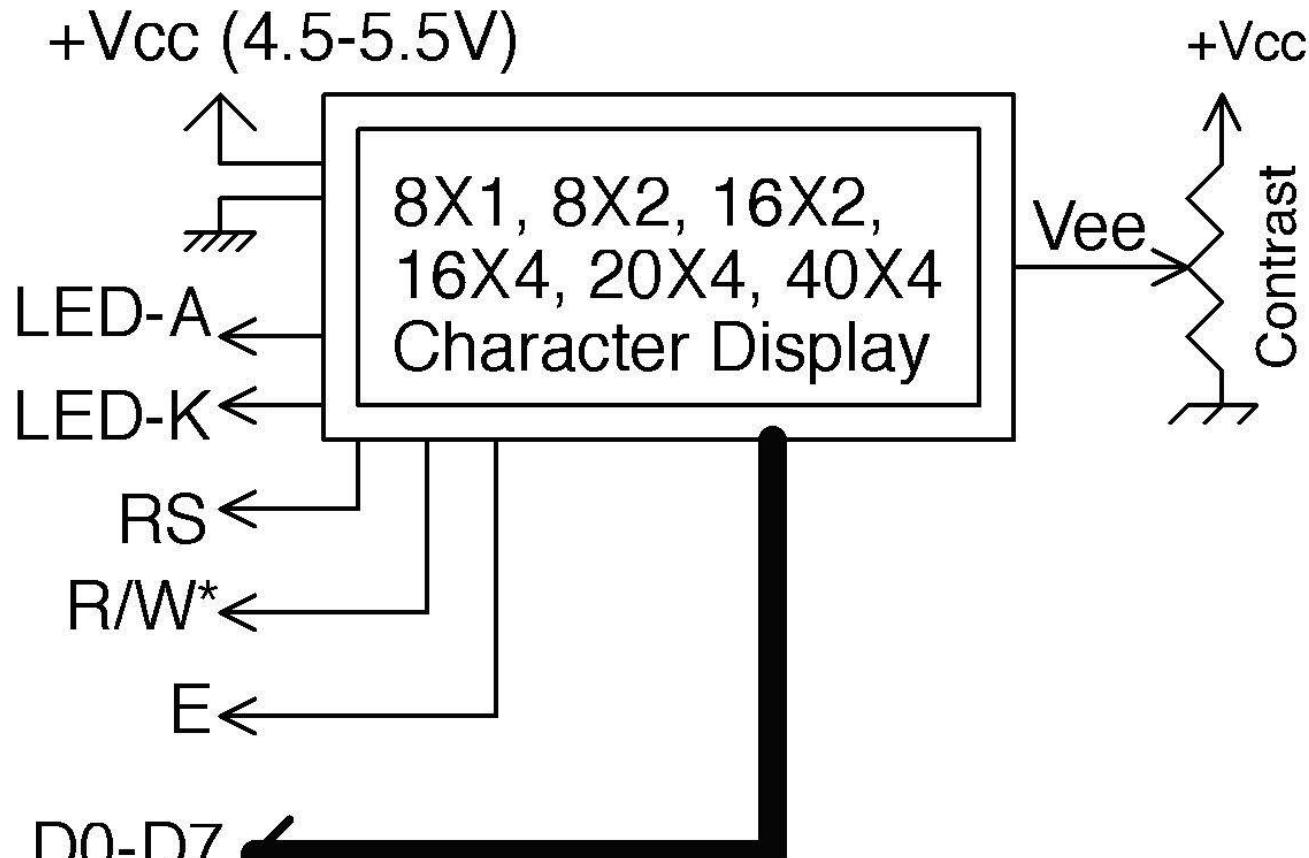
Digital to Analog Conversion: Using PWM



Digital to Analog Conversion: Using R-2R Ladder DAC

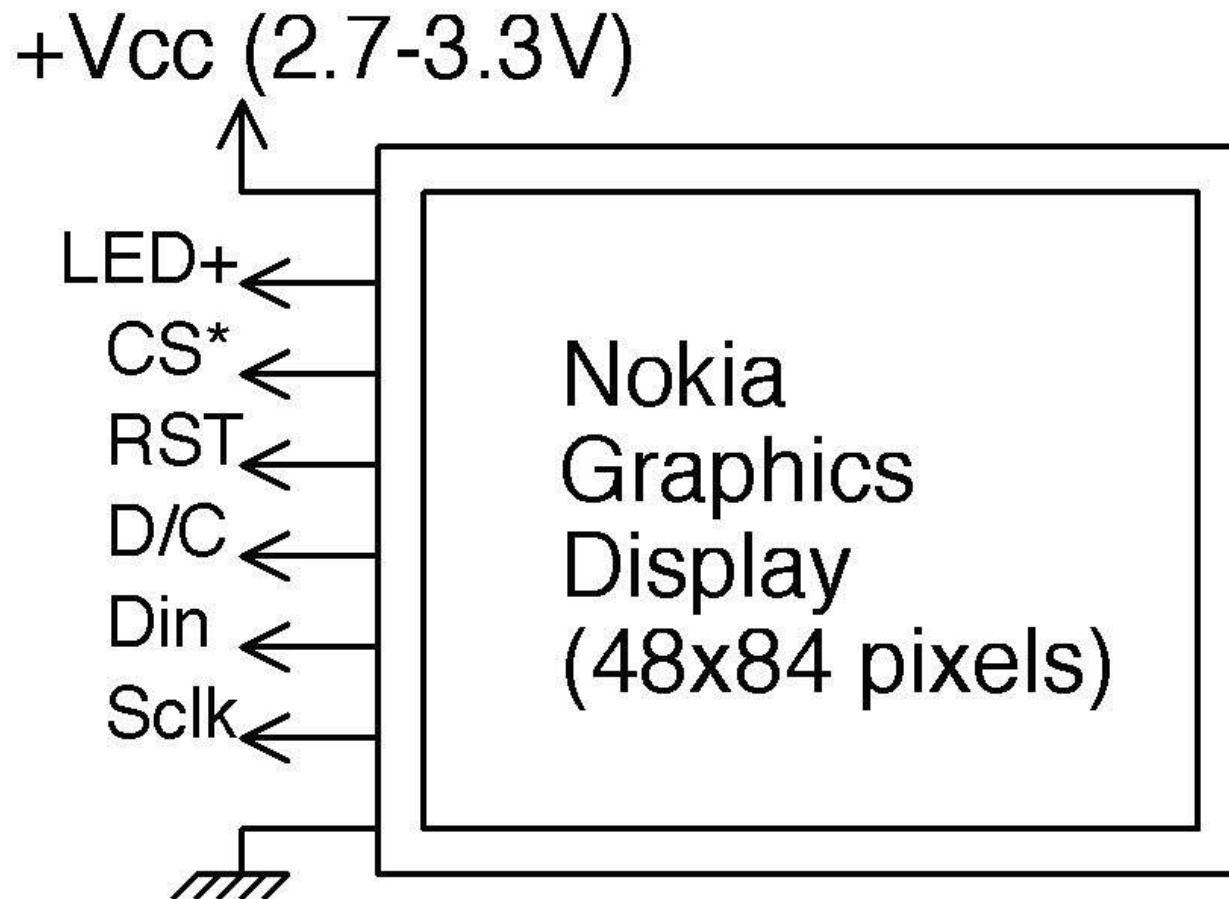


Liquid Crystal Displays



Hitachi HD LCD Controller (5x8 dot-matrix)

Liquid Crystal Displays



Buzzer and Speaker

- **Buzzer: In-built driver**
- **External driver: Tuned circuit actuated using timer.**
- **Speaker: Transistor switch and Timer**
- **DAC + Audio amplifier**

Ultrasonic

- **Piezoelectric element with a strong resonance**

Haptic

- DC Motor with eccentric load



Thank you!

Introduction to Embedded System Design

Getting acquainted with GIT, CCS Installation and Embedded C

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Version Control System: GIT

Software link

- <https://git-scm.com/downloads>
- Follow normal installing steps

Initial configuration of user

- git config --global user.name "Username"
- git config --global user.email "email.id@domain.com"

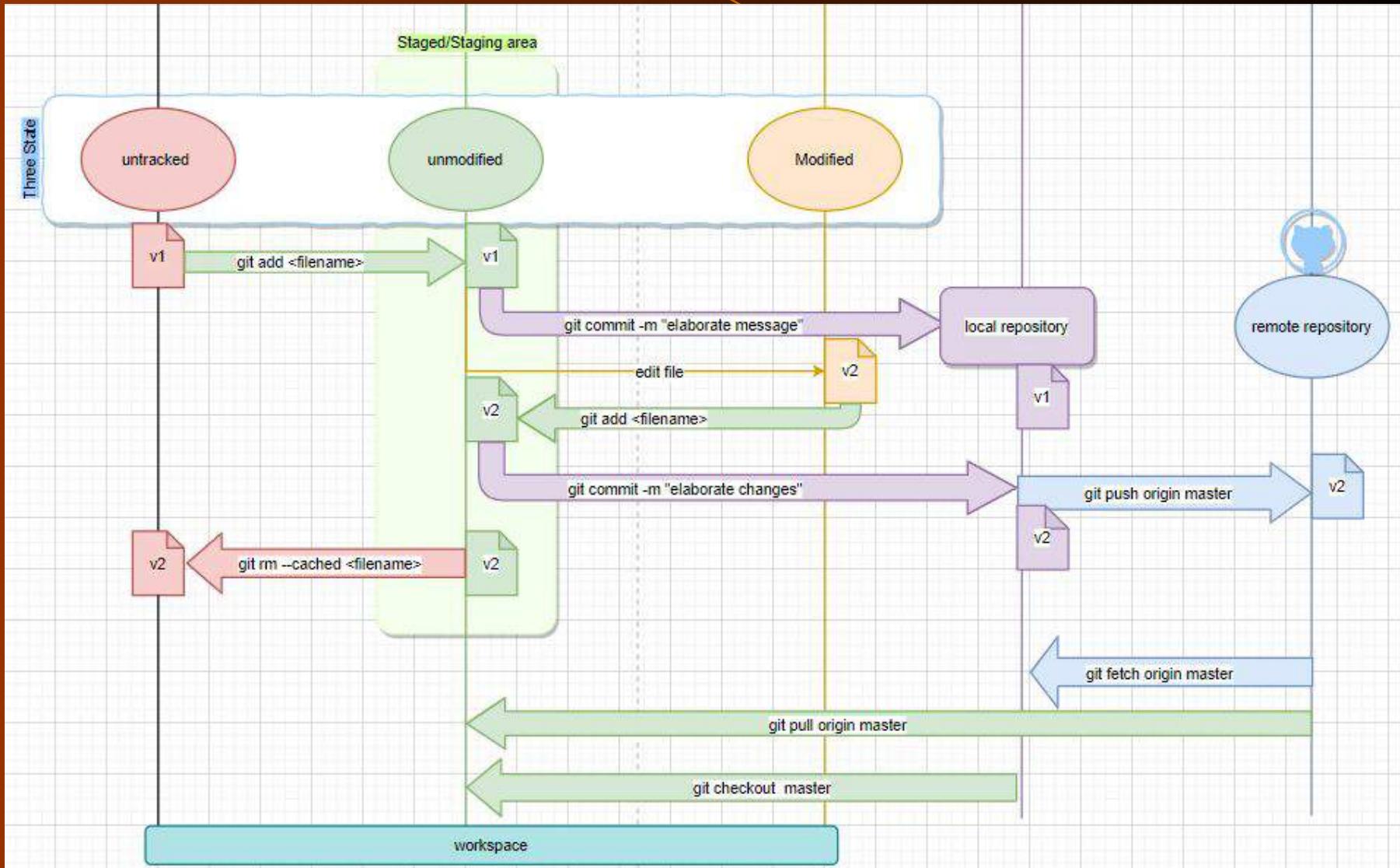
Create Repositories

- git init
or
- git clone <url>
 - For example: git clone https://github.com/ticepd/EmbSysDesign_NPTEL_Course.git

Handling changes

- git add <file>
- git commit -m “elaborate changes message”
- git log
- git log --follow <file>

Basic workflow of GIT



Synchronize changes

- git fetch
 - Downloads all history from the remote tracking branches.
- git pull
 - Updates your current local working branch with all new commits from the corresponding remote branch on GitHub.

Clone remote repository for this series

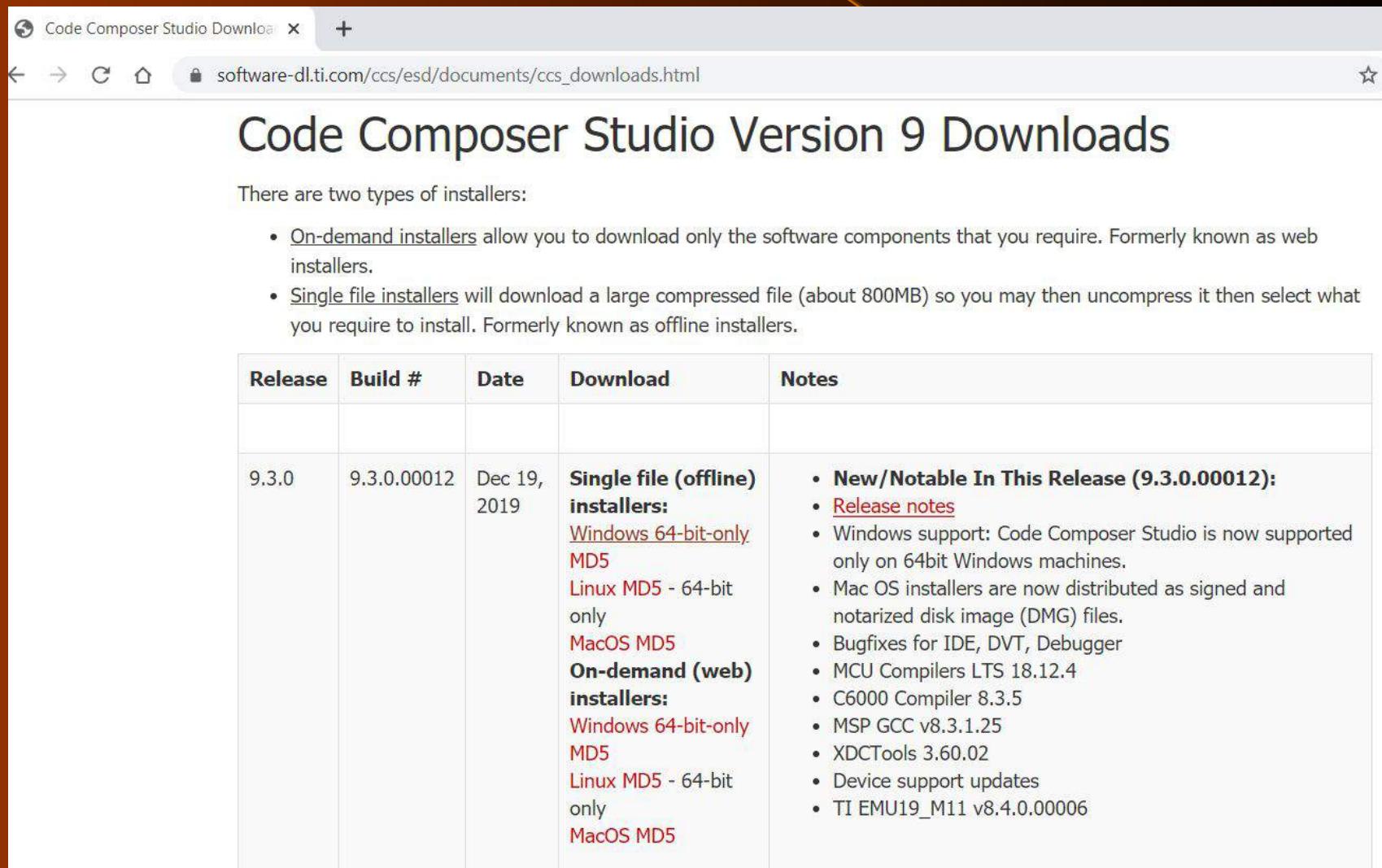
git clone
https://github.com/ticepd/EmbSysDesign_NPTEL_Course.git

Download files directly

https://github.com/ticepd/EmbSysDesign_NPTEL_Course/archive/master.zip

Setting up CCS

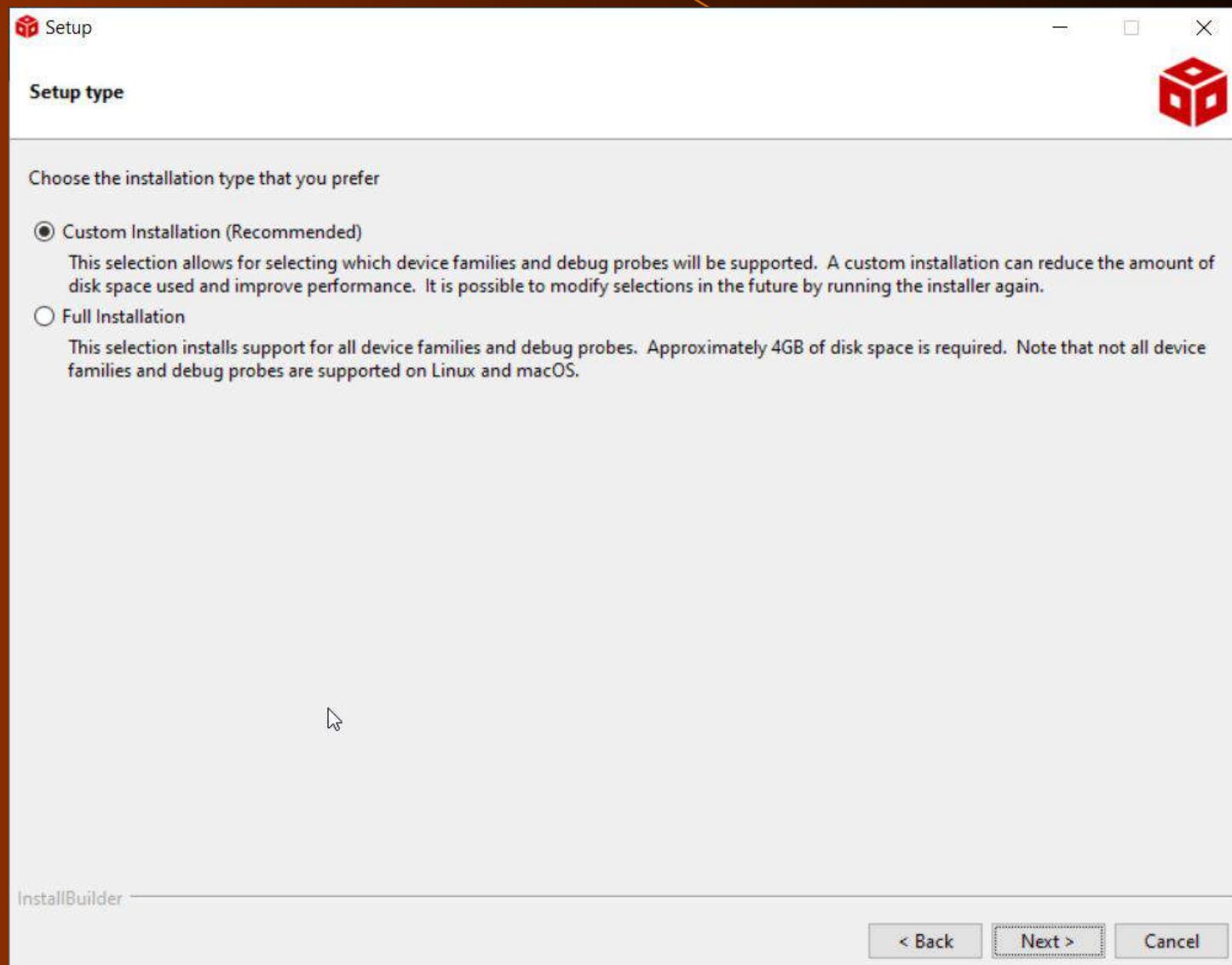
Setting up CCS : Download page



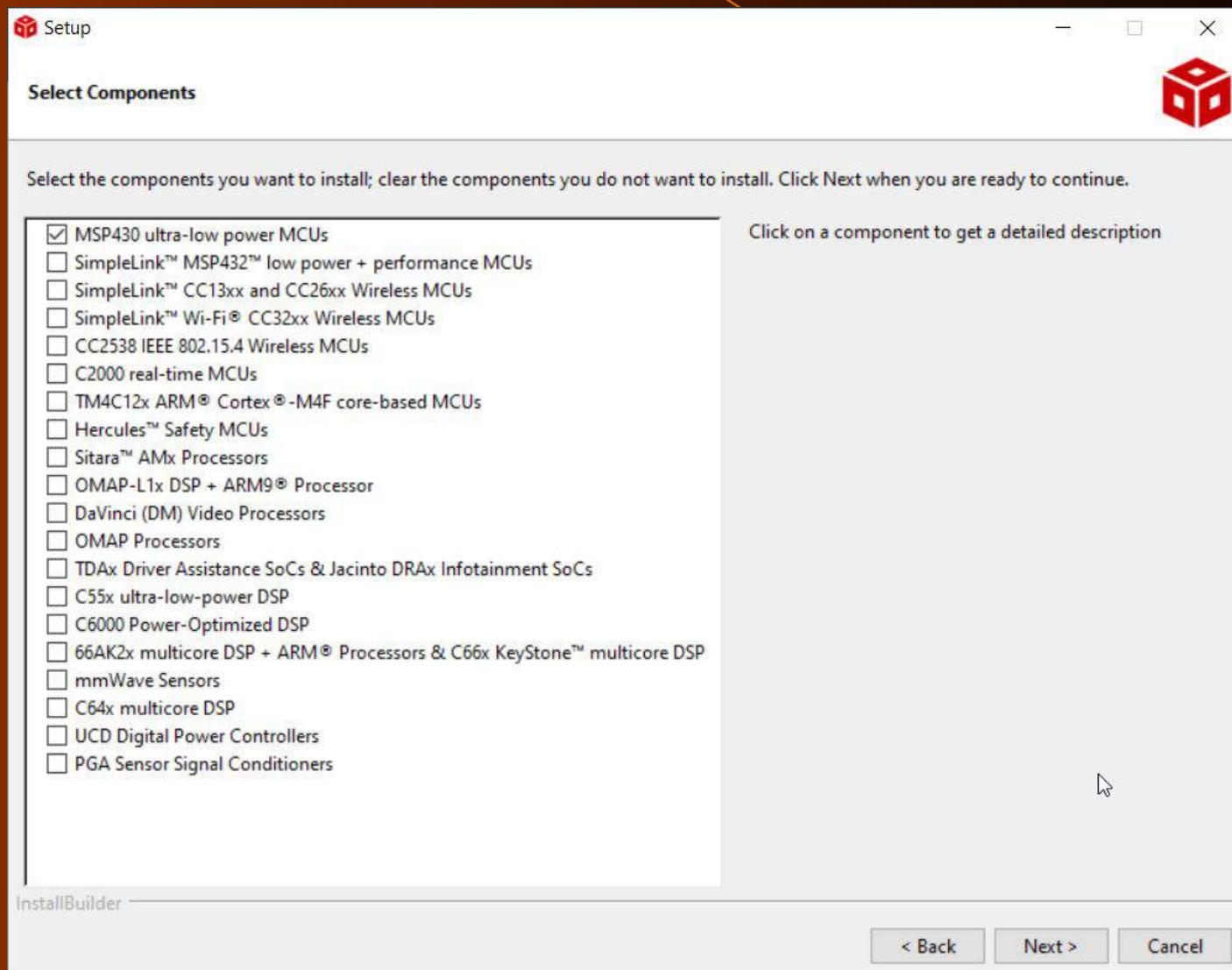
The screenshot shows a web browser window with the title "Code Composer Studio Download" and the URL "software-dl.ti.com/ccs/esd/documents/ccs_downloads.html". The main content is titled "Code Composer Studio Version 9 Downloads". It states that there are two types of installers: On-demand installers and Single file installers. A table lists the available releases, their build numbers, download dates, and notes. The first release listed is 9.3.0, build 9.3.0.00012, released on Dec 19, 2019. The notes for this release mention new/Notable features, release notes, Windows support (now supported only on 64bit Windows machines), Mac OS installers as signed and notarized disk image (DMG) files, bugfixes for IDE, DVT, Debugger, MCU Compilers LTS 18.12.4, C6000 Compiler 8.3.5, MSP GCC v8.3.1.25, XDCTools 3.60.02, Device support updates, and TI EMU19_M11 v8.4.0.00006.

Release	Build #	Date	Download	Notes
9.3.0	9.3.0.00012	Dec 19, 2019	Single file (offline) installers: Windows 64-bit-only MD5 Linux MD5 - 64-bit only MacOS MD5 On-demand (web) installers: Windows 64-bit-only MD5 Linux MD5 - 64-bit only MacOS MD5	<ul style="list-style-type: none">New/Notable In This Release (9.3.0.00012):Release notesWindows support: Code Composer Studio is now supported only on 64bit Windows machines.Mac OS installers are now distributed as signed and notarized disk image (DMG) files.Bugfixes for IDE, DVT, DebuggerMCU Compilers LTS 18.12.4C6000 Compiler 8.3.5MSP GCC v8.3.1.25XDCTools 3.60.02Device support updatesTI EMU19_M11 v8.4.0.00006

Setting up CCS



Setting up CCS : choose MSP430



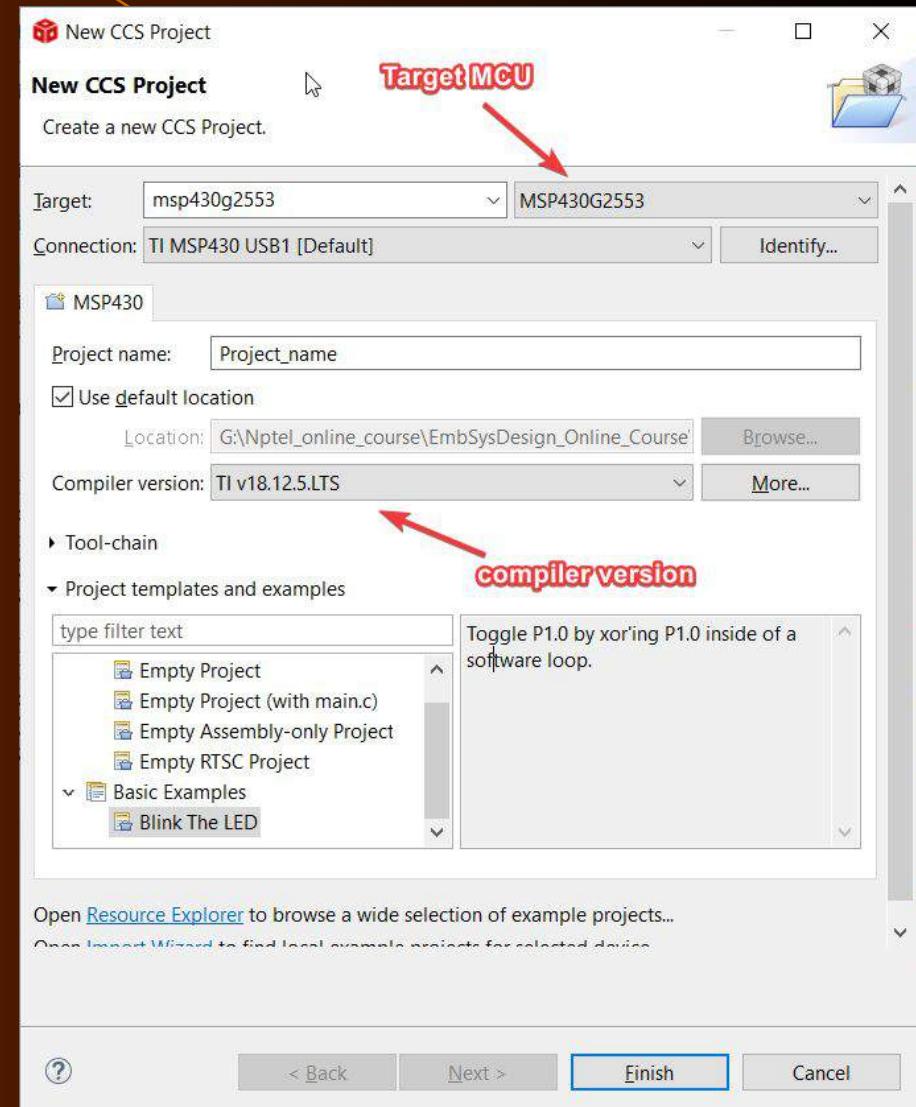
Import project in CCS after cloning from Github

- On starting CCS for the first time, Initial projects (provided on Github and as shown in upcoming videos) will not be shown inside CCS project explorer.
- To import those existing projects from git cloned path to CCS, Proceed with following steps inside CCS,
 - Go to File > Import.
 - General > Existing projects into workspace
 - Select root directory to
Software\Examples_Msp430G2553_LunchBox in cloned directory
 - Refresh > Select All
 - Finish

Setting up a New Project for MSP430G2553

Navigate using following steps:-

- File > New > CCS project



Automatically generated settings with project

Properties for Exp02_LunchBox_HelloBlink

type filter text

> Resource
General
Build
MSP430 Compiler
Processor Options
Optimization
Include Options
ULP Advisor
Advice Options
Predefined Symbols
> Advanced Options
> MSP430 Linker
> MSP430 Hex Utility
Debug
Git
Project Natures

Predefined Symbols

Configuration: Debug Manage Configurations...

Pre-define NAME (--define, -D)
MSP430G2553

used during conditional compilation

Undefine NAME (--undefine, -U)
INLINE

LLVM Optimization Level (--llopt)

Quick Access

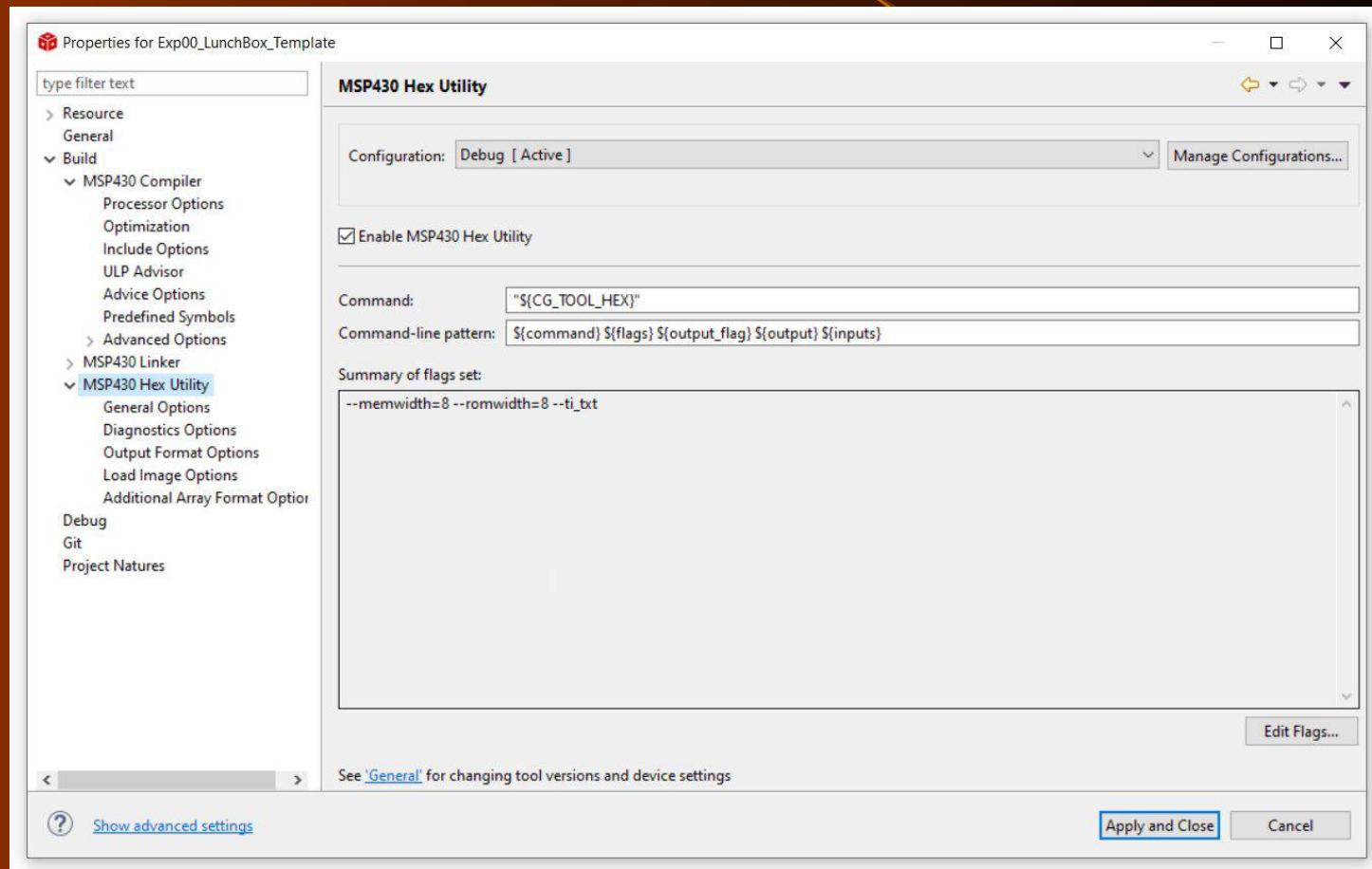
msp430.h msp430g2553.h

1418
1419 #elif defined (_MSP430G2453_)
1420 #include "msp430g2453.h"
1421
1422 #elif defined (_MSP430G2553_)
1423 #include "msp430g2553.h"
1424
1425 #elif defined (_MSP430G2203_)
1426 #include "msp430g2203.h"

Target MSP430G2553

BSL settings in CCS

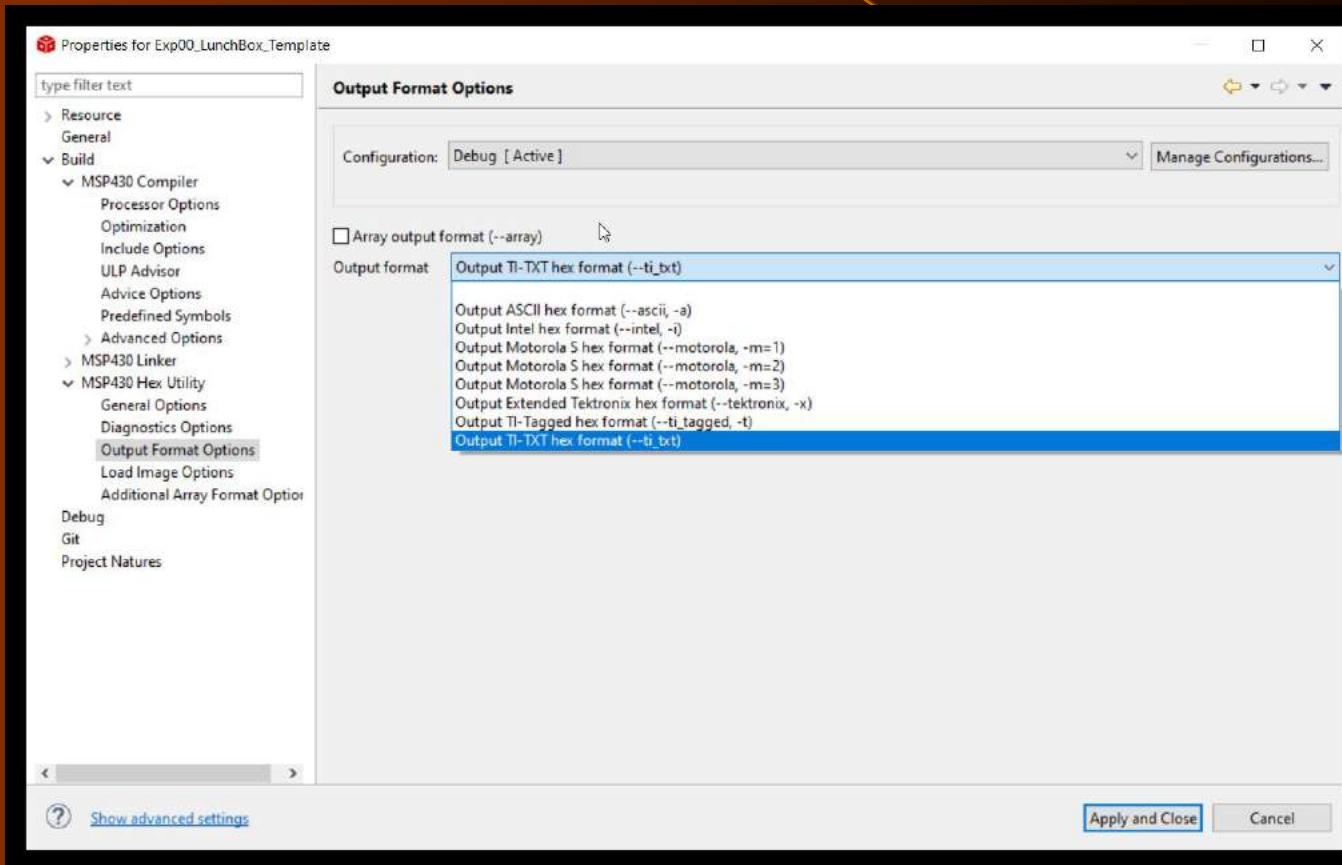
Step 1



- Project Properties > Build > MSP430 Hex Utility > Enable MSP430 Hex Utility

BSL settings in CCS

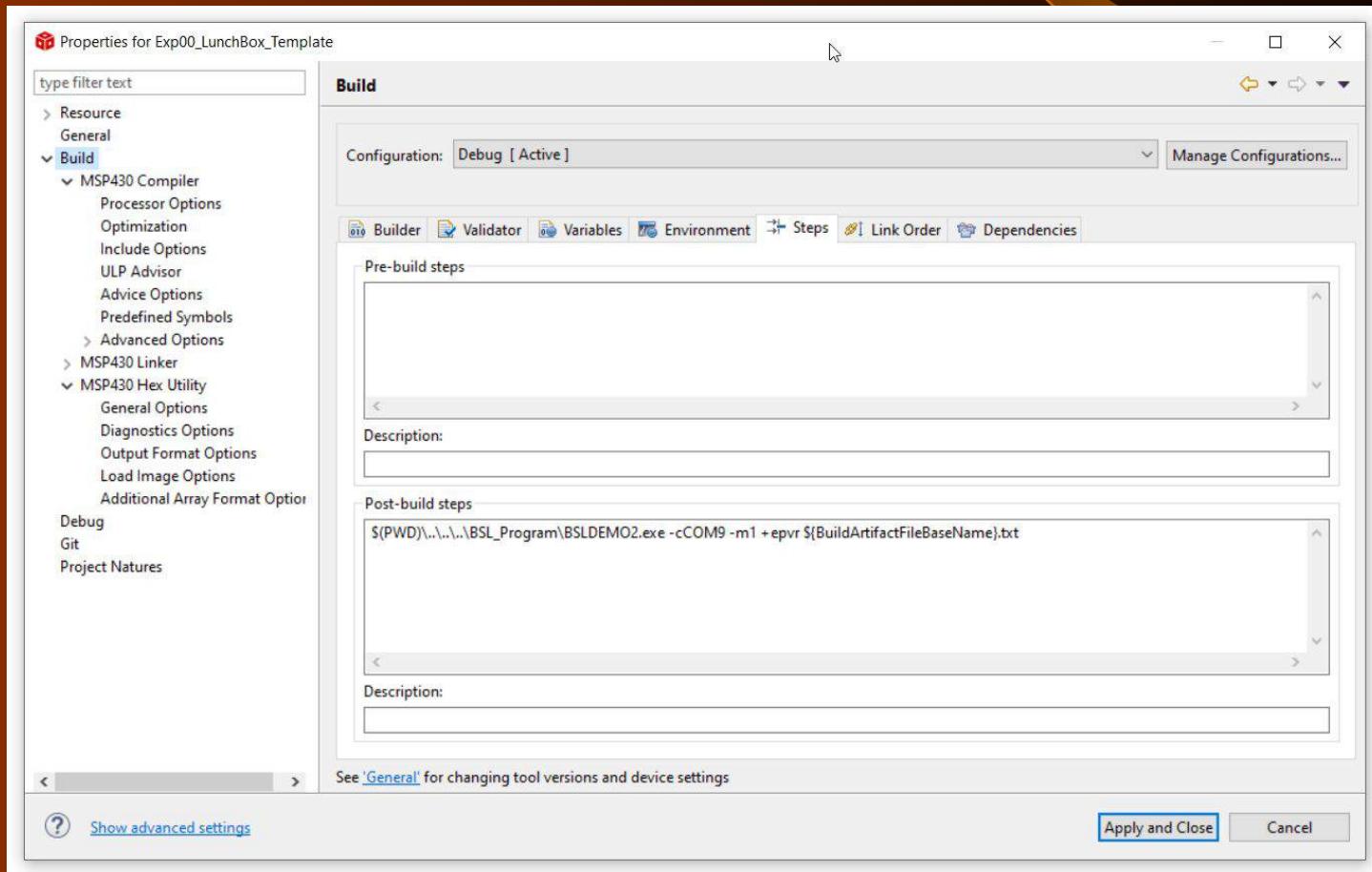
Step 2



- Project Properties > Build > MSP430 Hex Utility > Output Format Options > Output TI-TXT hex format

BSL settings in CCS

- Project Properties > Build > Steps tab > Post-build steps
- `$(PWD)\..\..\..\BSL_Program\BSLDEMO2.exe -cCOM9 -m1 +epvr ${BuildArtifactFileName}.txt`



MSP430 C/C++ compiler

The TI compiler accepts C and C++ code conforming to the International Organization for Standardization (ISO) standards for these languages. The compiler supports both the **1989 and 1999 versions of the C language** and the 2014 version of the C++ language.

Keywords in C

The MSP430 C/C++ compiler supports all of the standard C89 keywords, including const, volatile, and register. It also supports all of the standard C99 keywords, including inline and restrict. It also supports TI extension keywords __interrupt, and __asm. Some keywords are not available in strict ANSI mode.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

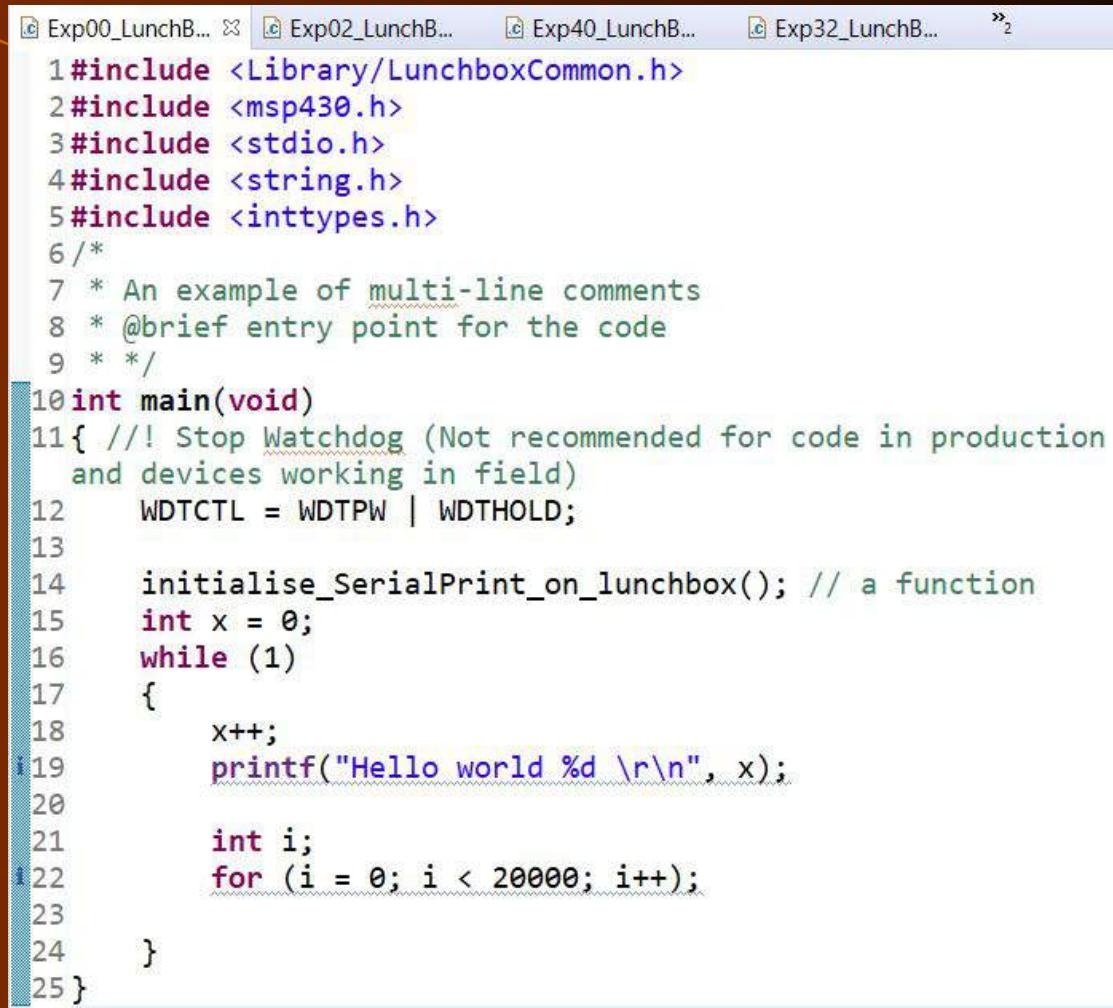
Main function

- Entry point
- Its compulsory.
- Good practice is not to exit it. (use while loop)

General main() code flow

- Configure WDT
- Set up the clock
- Configure ports and peripherals
- Enable Interrupts

Structure of Embedded C code



The image shows a screenshot of a code editor window titled "Exp00_LunchB...". The window displays a C program with syntax highlighting. The code includes multiple #include directives, multi-line comments, and a main function that initializes serial printing and enters a loop. A blue vertical bar highlights the first few lines of the code.

```
1 #include <Library/LunchboxCommon.h>
2 #include <msp430.h>
3 #include <stdio.h>
4 #include <string.h>
5 #include <inttypes.h>
6 /*
7 * An example of multi-line comments
8 * @brief entry point for the code
9 */
10 int main(void)
11{ //! Stop Watchdog (Not recommended for code in production
    and devices working in field)
12     WDTCTL = WDTPW | WDTHOLD;
13
14     initialise_SerialPrint_on_lunchbox(); // a function
15     int x = 0;
16     while (1)
17     {
18         x++;
19         printf("Hello world %d \r\n", x);
20
21         int i;
22         for (i = 0; i < 20000; i++);
23
24     }
25 }
```

Elements of Embedded Programming

Structure of the program:

A program generally has the following parts

- Library
- Main function
- Subroutines
- Interrupt Service Routines

C keywords provided to manipulate memory allocations

- Variable types (eg. int, char, uint8_t, bool, enum)
- Type modifiers
 - size manipulation (eg. short, long)
 - sign manipulation (eg. unsigned, signed)

C keywords provided to manipulate memory allocations

- Storage classes
 - Manipulate lifetime and scope
 - Examples
 - auto
 - It allocates local variable in stack memory
 - static
 - Data will persist till program execution
 - extern
 - Allow access outside current scope/file
 - register
 - Allocates memory in cpu register. It's not so common, compiler does it better

Variables qualifier

- `const` : Declaring a variable as a constant means that it's value cannot be modified by the program.
- `volatile`: By declaring a variable as volatile, the compiler gets to know that the value of the variable is susceptible to frequent changes (with no direct action of the program) and hence the compiler does not keep a copy of the variable in a register (like cache). This prevents the program to misinterpret the value of this variable.
- As a thumb rule: variables associated with input ports should be declared as volatile.

Data types in C

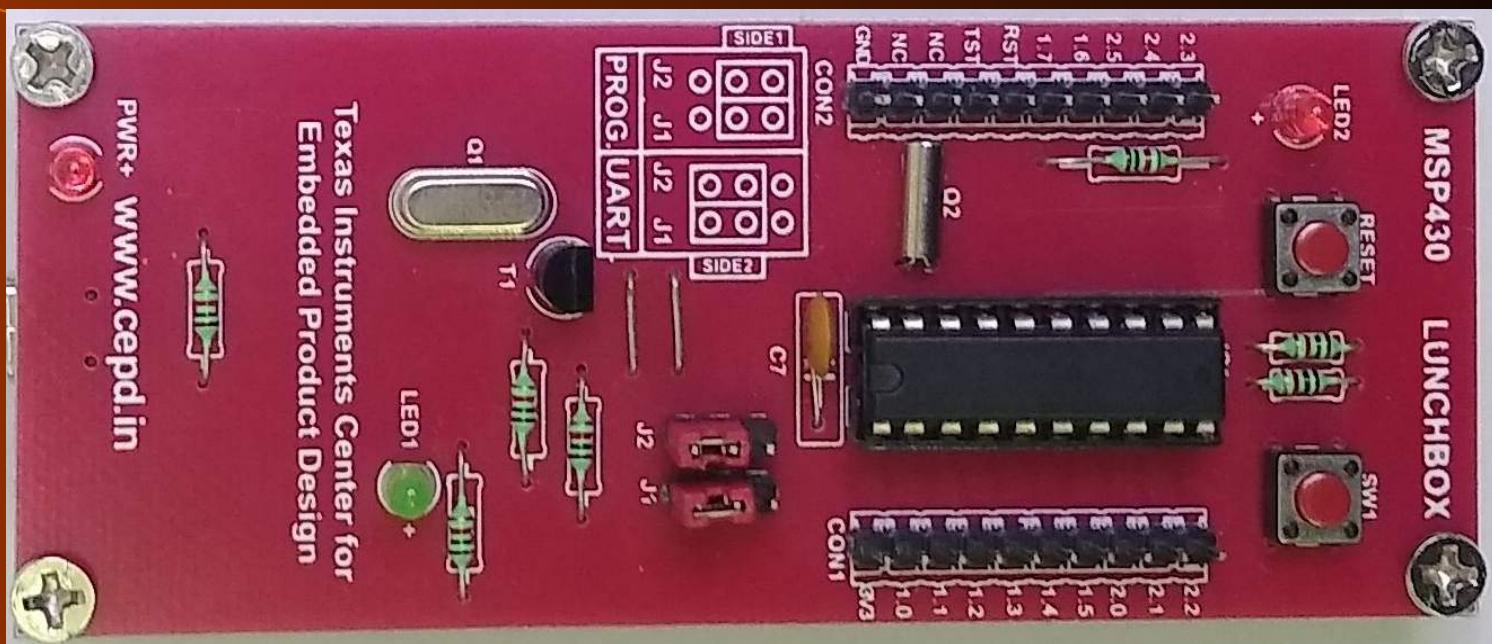
Type	Size	Alignment	Representation	Range	
				Minimum	Maximum
signed char	8 bits	8	Binary	-128	127
char	8 bits	8	ASCII	0 or -128 ⁽¹⁾	255 or 127 ⁽¹⁾
unsigned char	8 bits	8	Binary	0	255
bool (C99)	8 bits	8	Binary	0 (false)	1 (true)
_Bool (C99)	8 bits	8	Binary	0 (false)	1 (true)
bool (C++)	8 bits	8	Binary	0 (false)	1 (true)
short, signed short	16 bits	16	2s complement	-32 768	32 767
unsigned short	16 bits	16	Binary	0	65 535
int, signed int	16 bits	16	2s complement	-32 768	32 767
unsigned int	16 bits	16	Binary	0	65 535
long, signed long	32 bits	16	2s complement	-2 147 483 648	2 147 483 647
unsigned long	32 bits	16	Binary	0	4 294 967 295
long long, signed long long	64 bits	16	2s complement	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
unsigned long long	64 bits	16	Binary	0	18 446 744 073 709 551 615
enum	varies ⁽²⁾	16	2s complement	varies	varies
float	32 bits	16	IEEE 32-bit	1.175 494e-38 ⁽³⁾	3.40 282 346e+38
double	64 bits	16	IEEE 64-bit	2.22 507 385e-308 ⁽³⁾	1.79 769 313e+308
long double	64 bits	16	IEEE 64-bit	2.22 507 385e-308 ⁽³⁾	1.79 769 313e+308
function and data pointers	varies (see	16			

uintXX_t Data types in C

These data types (Example → `uint8_t`, `uint16_t`) are included in the header file - “inttypes.h”.

It's very useful when you do computations on bits, or on specific range of values (in cryptography, or image processing for example) because you don't have to "detect" the size of a long, or an unsigned int, you just "use" what you need. If you want 8 unsigned bits, use `uint8_t` like you did.

These are cross-platform. You can compile your program on a 32-bits or a 16-bits controller, and you won't have to change the types.



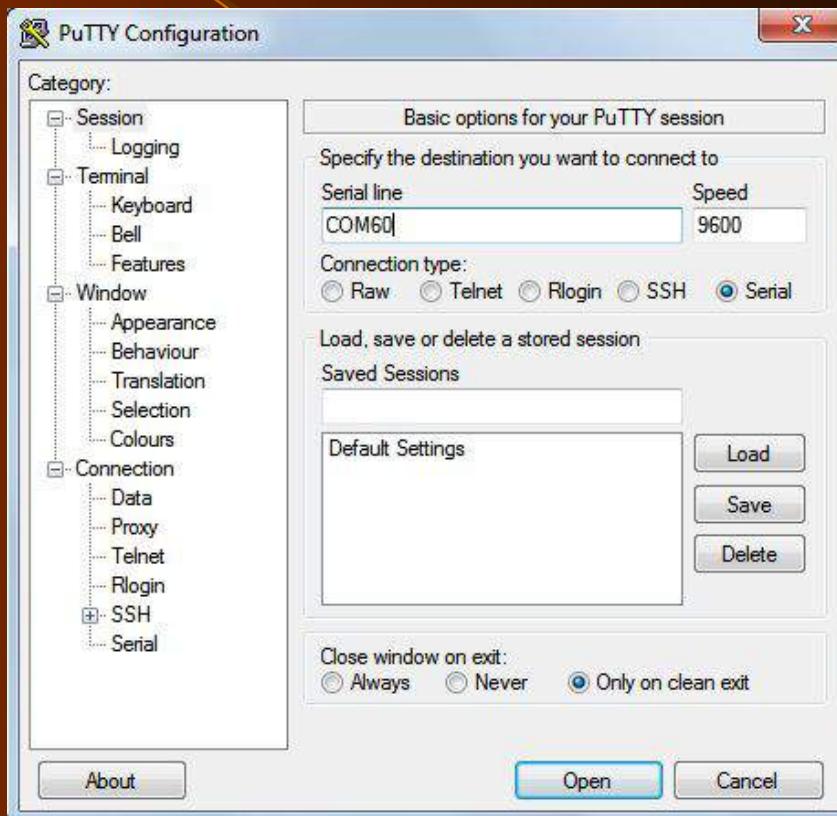
PWR+ www.cepd.in

Texas Instruments Center for
Embedded Product Design

Download and Install PuTTY

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application.

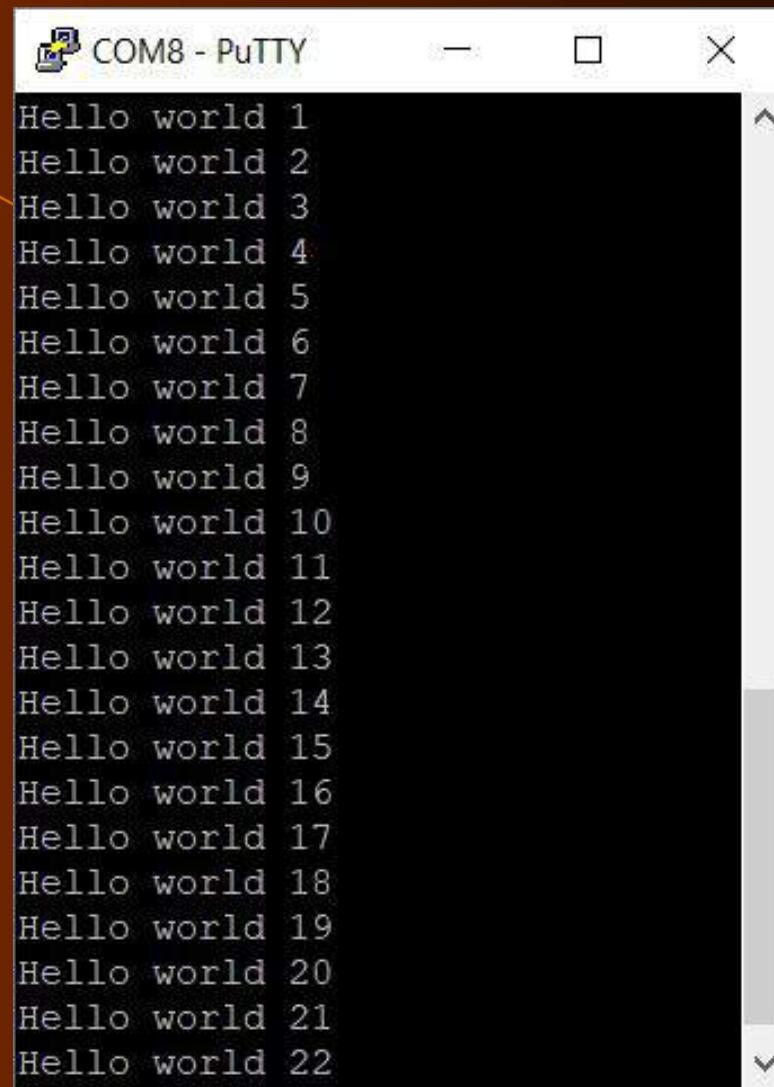
PuTTY can be used as a serial terminal for this Code Example.



Steps to invoke :-
double click, change
port number and click
open

Download link :-
<https://www.putty.org/>

Debugging using Serial Monitor



```
COM8 - PuTTY
Hello world 1
Hello world 2
Hello world 3
Hello world 4
Hello world 5
Hello world 6
Hello world 7
Hello world 8
Hello world 9
Hello world 10
Hello world 11
Hello world 12
Hello world 13
Hello world 14
Hello world 15
Hello world 16
Hello world 17
Hello world 18
Hello world 19
Hello world 20
Hello world 21
Hello world 22
```

Bit level access tricks on registers using C

- Set Bit:
P1DIR = 8; //0000 1000
P1DIR |= (1 << 2); //0000 0100
P1DIR |= (BIT7 + BIT6) // 1100 0000
→ Result 1100 1100
- Clear Bit:
P1DIR = 204; //1100 1100
P1DIR &= ~(1 << 3);
→ Result 1100 0100
- Flip Bit:
P1DIR = 8; //0000 1000
P1DIR ^= (1 << 5);
→ Result 0010 1000
- Get Bit:
P1DIR = 8; //0000 1000
if((P1DIR & (1 << 3)) > 0)
{
 // do this if the 3rd bit is set(1)
}
else
{
 // do this if the 3rd bit is not set(0)
}

Example of Embedded C: demo of Bit Manipulation

```
COM8 - PuTTY

x value      000000010
y value      0000000100
z value      000001000

Bitwise operators in C

value of a & b      000000100
value of a | b      001100100
value of a<<1    010001000
value of a^b        001100000
value of (1<<3)    000001000
value of a|(1<<4)  001010100
```

```
c Exp41_LunchBox_Introduction_to_EMBEDDED_C.c ✘

1 #include <msp430.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <inttypes.h>
5 #include <Library/LunchboxCommon.h>
6 int main(void)
7{ //! Stop Watchdog (Not recommended for code in production
8 //and devices working in field)
9 WDTCTL = WDTPW | WDTHOLD;

10
11 initialise_SerialPrint_on_lunchbox(); // a function
12 while (1)
13 {
14     uint8_t x = 2;
15     uint8_t y = 4;
16     uint8_t z = 8;
17     printf("\r\n x value \t");
18     decToBinary(x);
19     printf("\r\n y value \t");
20     decToBinary(y);
21     printf("\r\n z value \t");
22     decToBinary(z);
23     printf("\r\r\r\r\r");
24     printf("\r\r Bitwise operators in C \r\r");
25     uint8_t a = 0b01000100; //68 or 0x44
26     uint8_t b = 0b00100100; //36 or 0x24
27     printf("\r\r value of a & b \t");
28     decToBinary(a & b);
29     printf("\r\r value of a | b \t");
30     decToBinary(a | b);
31     printf("\r\r value of a<<1 \t");
32     decToBinary(a << 1);
33     printf("\r\r value of a^b \t");
34     decToBinary(a ^ b);
35     printf("\r\r value of (1<<3) \t");
36     decToBinary(1 << 3);
37     printf("\r\r value of a|(1<<4) \t");
38     decToBinary(a | (1 << 4));
39     printf("\r\r\r\r\r");
40     unsigned long delay;
41     for (delay = 0; delay < 60000; delay++);
42 }
43 }
```

Blink led example using msp430.h

```
1 #include <msp430.h>
2 /*@brief entry point for the code*/
3 void main(void)
4 {
5     WDTCTL = WDTPW | WDTHOLD;
6     //(*(volatile unsigned int *) 0x0120) = 0x5A00 | 0x0080;
7     /* ! Stop Watchdog (Not recommended for code in production
8      and devices working in field)
9      */
10
11    P1DIR |= BIT7;
12    //    (*(volatile unsigned char *) 0x0022) |= 0x80;
13    // P1.7 (Red LED)
14
15    volatile unsigned long i;
16    while (1)
17    {
18
19        //P1OUT |= BIT7;
20        P1OUT |= 0x80;           //Red LED -> ON
21        //    (*(volatile unsigned char *) 0x0021) |= 0x80;
22        for (i = 0; i < 10000; i++)
23            ; //delay
24
25        //P1OUT &=~ BIT7;
26        P1OUT &= ~0x80;          //Red LED -> OFF
27        //    (*(volatile unsigned char *) 0x0021) &= ~0x80;
28        for (i = 0; i < 10000; i++)
29            ; //delay
30    }
31 }
```

Blink led example using direct write to port location

```
c Exp02_LunchBox_HelloBlink_without_msp430_h.c ✘
1 // #include <msp430.h>
2 /*@brief entry point for the code*/
3 void main(void)
4 {
5 //  WDTCTL = WDTPW | WDTHOLD;
6 (*(volatile unsigned int *) 0x0120) = 0x5A00 | 0x0080;
7 /* ! Stop Watchdog (Not recommended for code in production
8 and devices working in field)
9 */
10
11 //P1DIR |= BIT7;
12 (*(volatile unsigned char *) 0x0022) |= 0x80;
// P1.7 (Red LED)
13
14
15 volatile unsigned long i;
16 while (1)
17 {
18
19 //P1OUT |= BIT7;
20 // P1OUT |= 0x80; //Red LED -> ON
21 (*(volatile unsigned char *) 0x0021) |= 0x80;
//Red LED -> ON
22
23 for (i = 0; i < 10000; i++)
24 ; //delay
25
26 //P1OUT &=~ BIT7;
27 //P1OUT &= ~0x80; //Red LED -> OFF
28 (*(volatile unsigned char *) 0x0021) &= ~0x80;
29
30 for (i = 0; i < 10000; i++)
31 ; //delay
32 }
```



Thank you!

Introduction to Embedded System Design

MSP430 Digital I/O

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

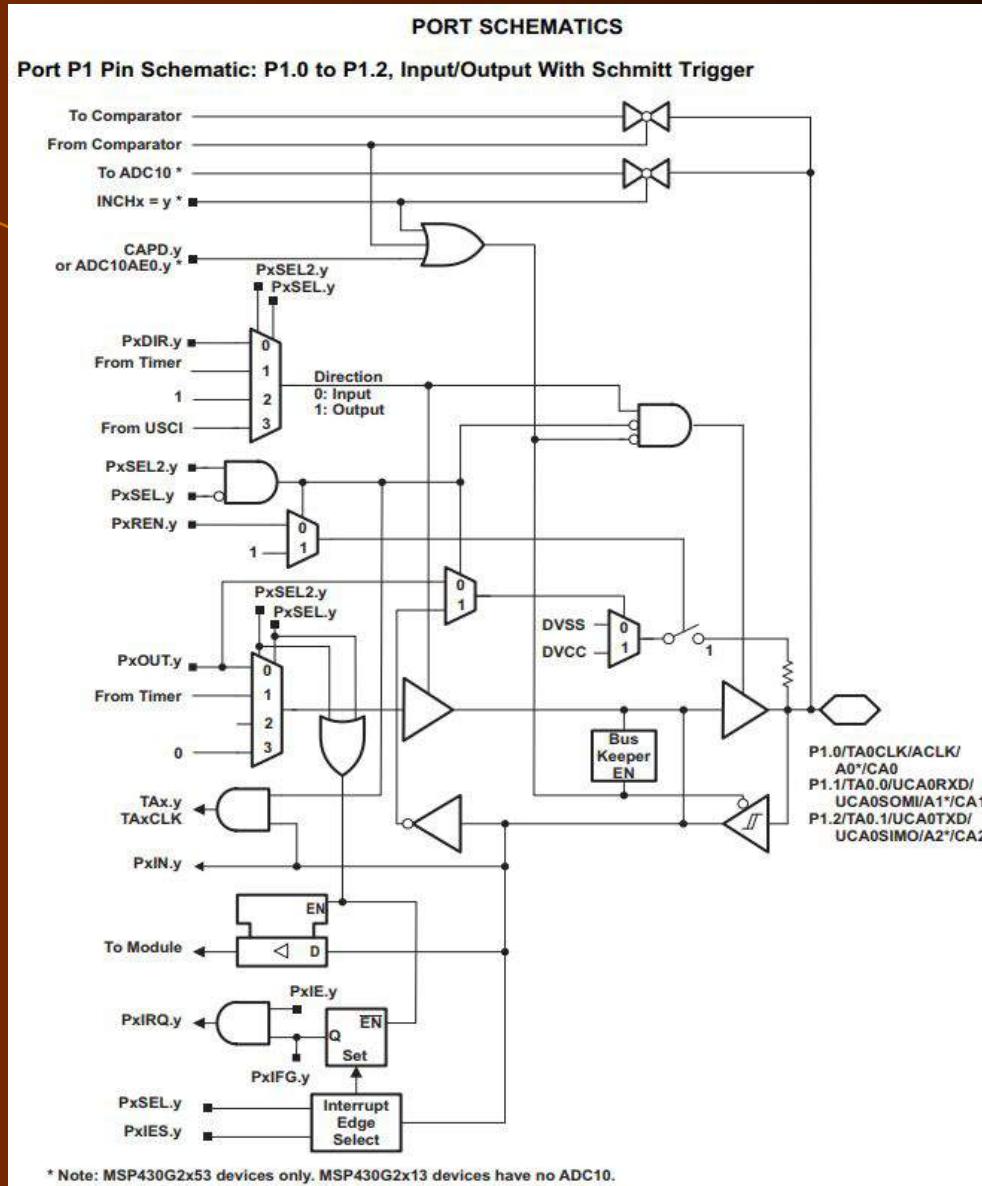
Electrical Engineering Department

Indian Institute of Technology,
Jammu

Digital I/O

- MSP430G2553 (20 pin) has two Ports P1 and P2.
- MSP430G2553 (28 pin) has three ports P1,P2,P3.
- Each Port has 8 I/O pins.
- Any I/O can be made input or output, and can be individually read or written to.
- Individually configurable interrupts on P1 and P2.
- Choice of individually configuring pull-up or pull down resistors.
- The ports are configured by several 8 bit Peripheral Registers.

Port Schematics



Port Schematics

Settings for selecting the function of a given pin is provided in respective datasheet. Example is provided here:-

Table 16. Port P1 (P1.0 to P1.2) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ⁽¹⁾				
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x=1 ⁽²⁾	CAPD.y
P1.0/ TA0CLK/ ACLK/ A0 ⁽²⁾ / CA0/ Pin Osc	0	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.TACLK	0	1	0	0	0
		ACLK	1	1	0	0	0
		A0	X	X	X	1 (y = 0)	0
		CA0	X	X	X	0	1 (y = 0)
		Capacitive sensing	X	0	1	0	0
P1.1/ TA0.0/ UCA0RXD/ UCA0SOMI/ A1 ⁽²⁾ / CA1/ Pin Osc	1	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.0	1	1	0	0	0
		TA0.CCI0A	0	1	0	0	0
		UCA0RXD	from USCI	1	1	0	0
		UCA0SOMI	from USCI	1	1	0	0
		A1	X	X	X	1 (y = 1)	0
		CA1	X	X	X	0	1 (y = 1)
		Capacitive sensing	X	0	1	0	0
P1.2/ TA0.1/ UCA0TXD/ UCA0SIMO/ A2 ⁽²⁾ / CA2/ Pin Osc	2	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.1	1	1	0	0	0
		TA0.CCI1A	0	1	0	0	0
		UCA0TXD	from USCI	1	1	0	0
		UCA0SIMO	from USCI	1	1	0	0
		A2	X	X	X	1 (y = 2)	0
		CA2	X	X	X	0	1 (y = 2)
		Capacitive sensing	X	0	1	0	0

(1) X = don't care

(2) MSP430G2x53 devices only

Digital I/O Registers

The MSP430 communicates with the Digital I/O Peripherals through a set of 8 bit Peripheral Registers:

- PxDIR
- PxIN
- PxOUT
- PxREN
- PxSEL and PxSEL2

PxDIR

- ‘P’ stands for port
- ‘x’ stands for port number (Upto 3 ports available on MSP430G2553, based on package selected)
- It is an eight bit register. It is used to define whether a pin is to be used as an input or as an output.
- If value of a bit in PxDIR register is ‘0’, it is set as input.
- If value of a bit in PxDIR register is ‘1’, it is set as output.
- Default value of PxDIR register for all ports is ‘0’.

PxIN

- This eight bit register stores the value of input on a port. Based on the value of logic low and logic high for MSP430 MCU, the voltage value read on the input pin will be classified as logic low or logic high.
- The ‘0’ value of a bit in PxIN register indicates the input value of bit as low.
- The ‘1’ value of a bit in PxIN register indicates the input value of bit as high.

For reading the value of a GPIO, set the pin as input using PxDIR register then read the value of bit in PxIN register.

PxOUT

- This eight bit register sets the digital value on a port.
- Setting the value of a bit as ‘0’ in PxOUT register sets the output value of bit as low.
- Setting the value of a bit as ‘1’ in PxOUT register sets the output value of bit as high.

For setting the value of a GPIO, set the pin as output using PxDIR register then set the value of bit in PxOUT register.

PxREN

- This eight bit register enables the pullup or pulldown resistor for a given pin.
- Setting the value of a bit as ‘0’ in PxREN register disables the pullup/pulldown resistor .
- Setting the value of a bit as ‘1’ in PxREN register enables the pullup/pulldown resistor.
- Once PxREN is set, PxOUT value is set to select between pullup or pulldown on pin. If PxOUT is set as ‘0’ then pin is pulled down. If PxOUT is set as ‘1’ then pin is pulled up.

PxSEL and PxSEL2

- Multiple functions are available on a single pin, PxSEL and PxSEL2 register are used to select function of a given pin.

PxSEL2	PxSEL	Pin Function
0	0	I/O function is selected.
0	1	Primary peripheral module function is selected.
1	0	Reserved. See device-specific data sheet.
1	1	Secondary peripheral module function is selected.

Bit Manipulation

In order to be able to configure the I/O and program the MSP430 we need to first understand some techniques to read/assess/configure/test/ manipulate individual bits in a register.

Bitwise Operation:-

1. OR (|)
2. AND (&)
3. XOR (^)
4. Shift Right (>>)
5. Shift Left (<<)
6. NOT (~)

We will learn some of these operations with examples in the following slides.

Setting outputs

The I/Os are independently configurable!

Example: For Port P1, suppose there are 8 LEDs connected on each pin. Suppose we have to turn on the LED on P1.3, without changing any configuration on the rest of the port. How will we do it?

Based on the logic that $x \mid 0 = x$ and $x \mid 1 = 1$, there are a number of ways to set a bit:

1. $P1OUT = P1OUT \mid \text{BIT3};$
2. $P1OUT \mid = \text{BIT3};$
3. $P1OUT = P1OUT \mid 0x08;$

Setting outputs

Now we have to turn off the LED on P1.3, without changing the output on the rest of the port. How will we do it?

Based on the logic that $x \& 0 = 0$ and $x \& 1 = x$, there are a number of ways to clear a bit:

1. $P1OUT = P1OUT \& (\sim BIT3);$
2. $P1OUT \&= \sim BIT3;$
3. $P1OUT = P1OUT \& 0xF7;$

Similarly for toggling the output on any pin:

$P1OUT = P1OUT \wedge BIT3;$

Reading inputs

Suppose there is a switch connected to P1.5 and we need to read the value of the pin.

Here is a snippet of code example for testing individual bits in a register:

```
if ((P1IN & BIT5) == 0)
    //to be done when P1.5 = 0
else
    // to be done when P1.5 = 1
```



Let's start programming!

Introduction to Embedded System Design

MSP430 Switch Interfacing

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

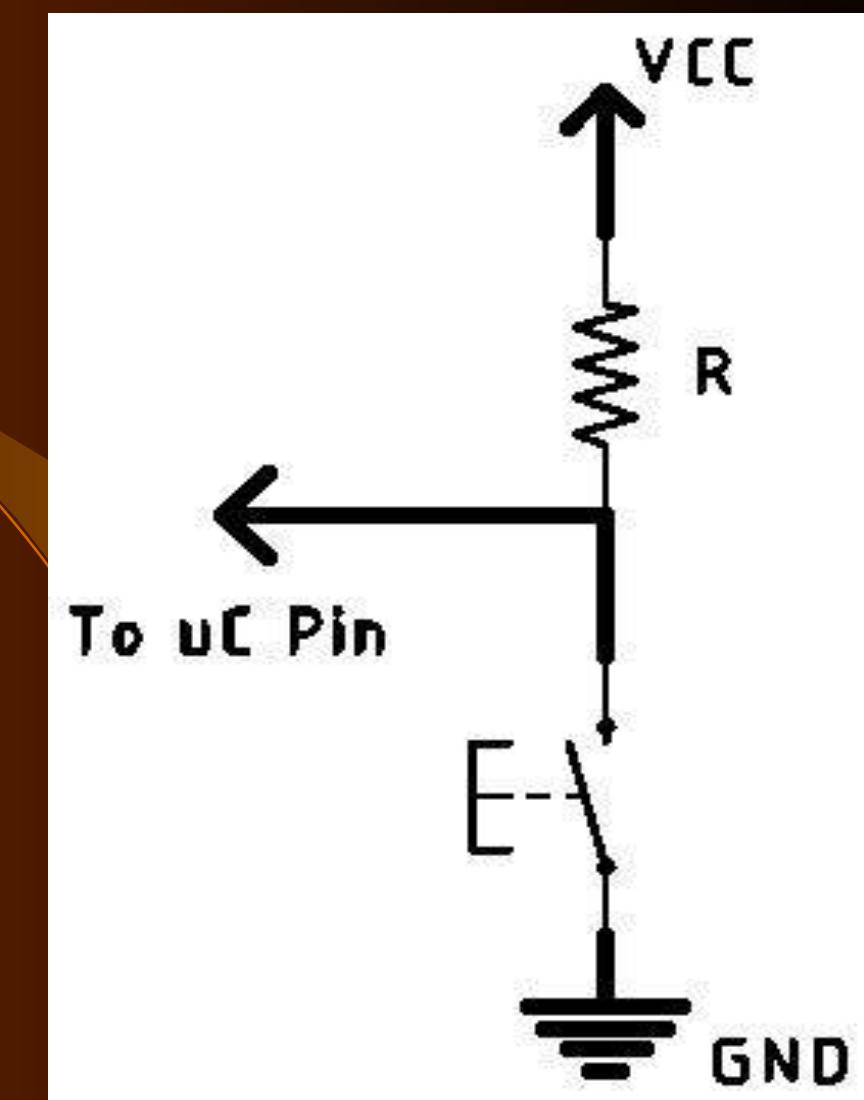
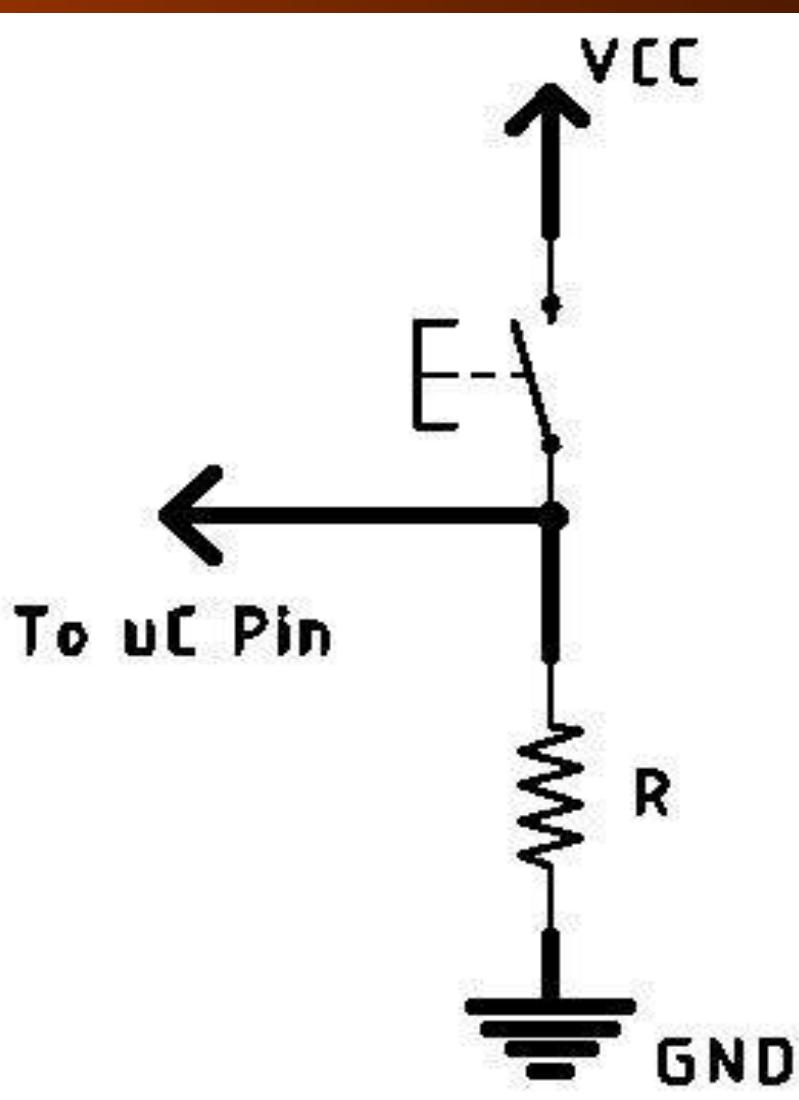
Indian Institute of Technology,
Jammu

Digital Switch Interfacing

Possible ways for a switch connection :

- Pull up configuration
- Pull down configuration

Digital Switch Interfacing



Digital Input Registers

The MSP430 registers required for switch inputs:

- PxDIR
- PxIN
- PxOUT
- PxREN

Reading inputs

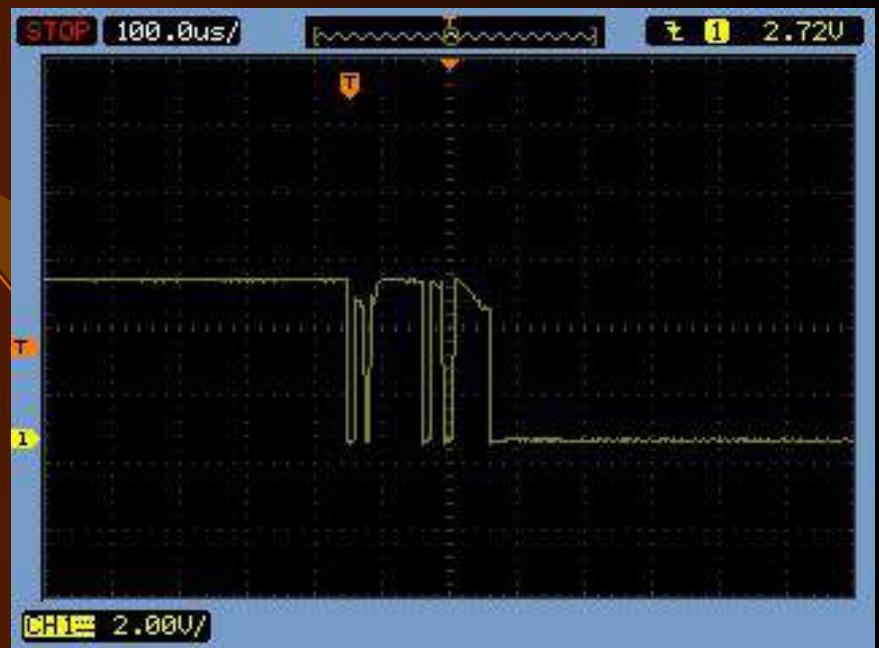
Suppose there is a switch connected to P1.3 in a pull up state (As is the case in LunchBox), and we need to read the value of the pin.

Here is a snippet of code example for testing individual bits in a register:

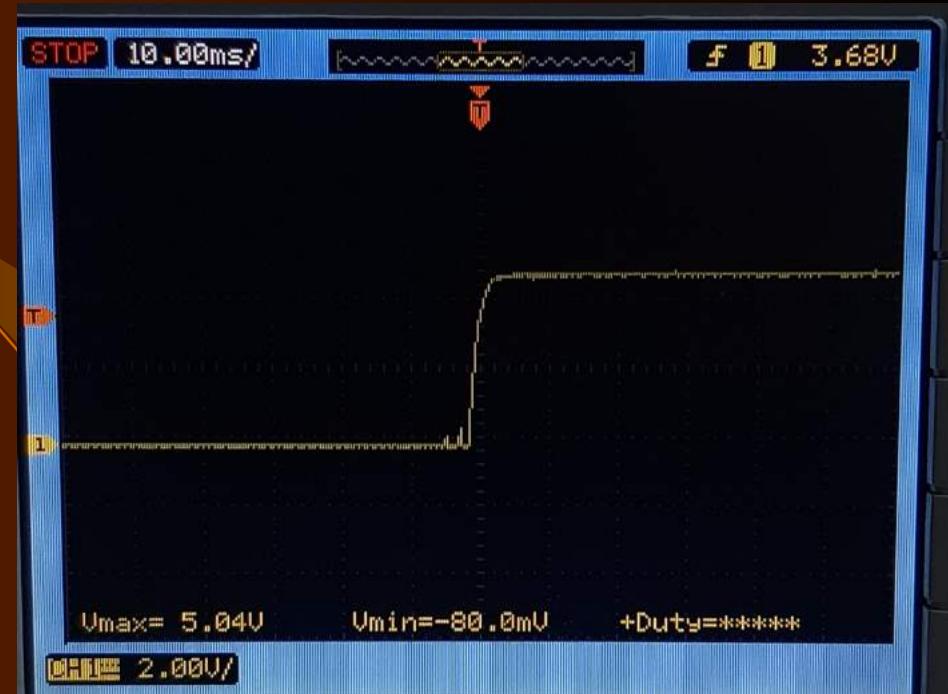
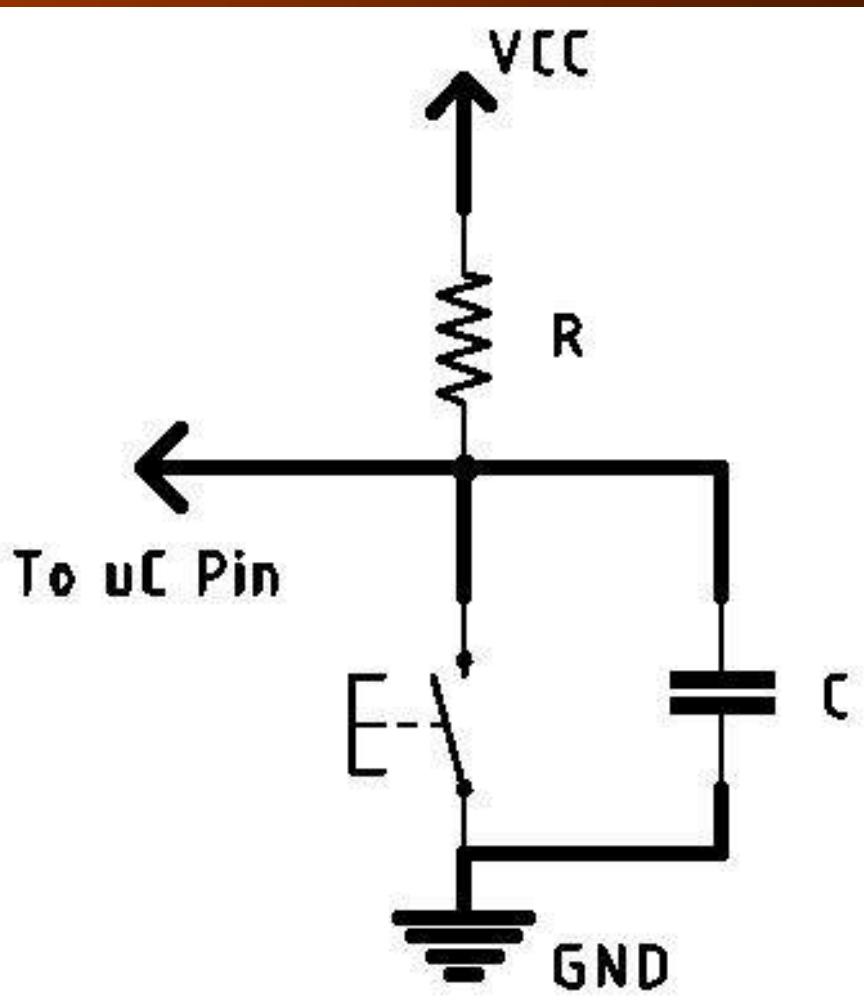
```
if ((P1IN & BIT3) == 0)
    //to be done when P1.3 = 0
else
    // to be done when P1.3 = 1
```

Switch Bouncing

- Common problem associated with the mechanical switches and relays.
- Made up of spring metals which are forced to contact each other by an actuator.
- While they collide each other there is a possibility of rebounding for some time before they make a stable contact.



Hardware Debouncing



Software Debouncing

- Provide a delay of ~20 milliseconds after the first change is detected.



Let's start programming!

Introduction to Embedded System Design

MSP430 Clock System and Reset

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

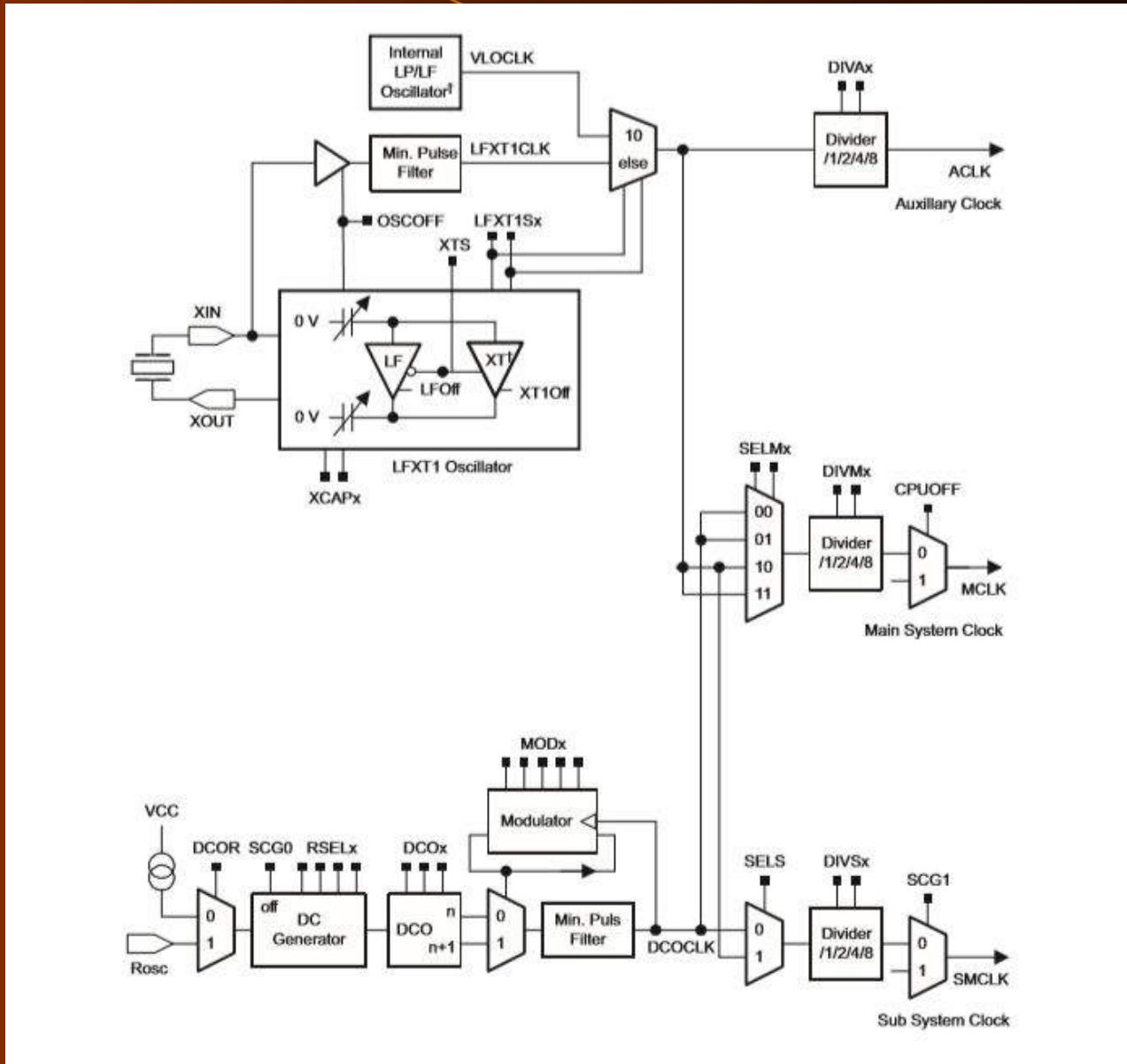
Badri Subudhi

Assistant Professor

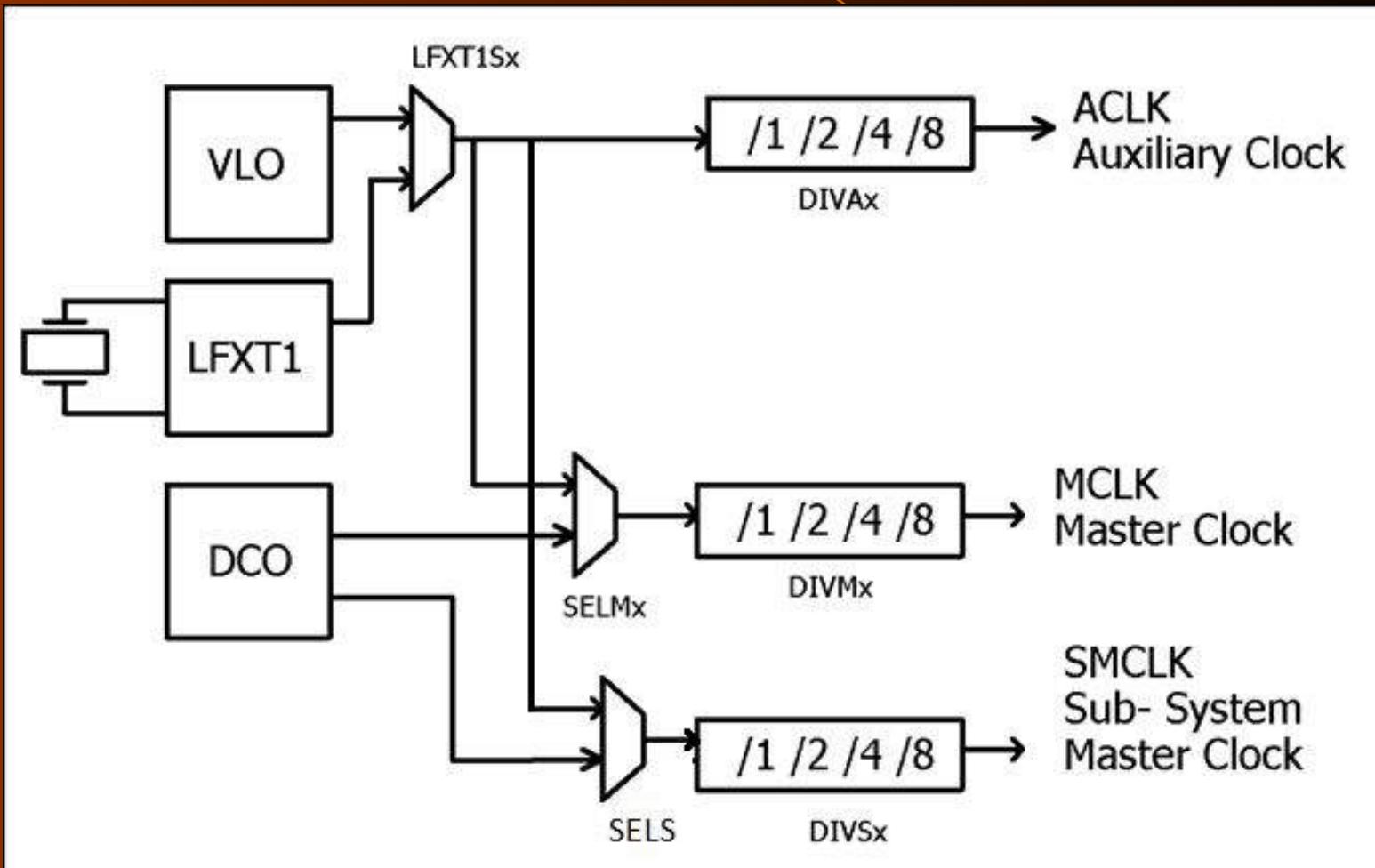
Electrical Engineering Department

Indian Institute of Technology,
Jammu

Basic Clock Module



Simplified Block Diagram of MSP430G2553 Clock System



Clock Sources

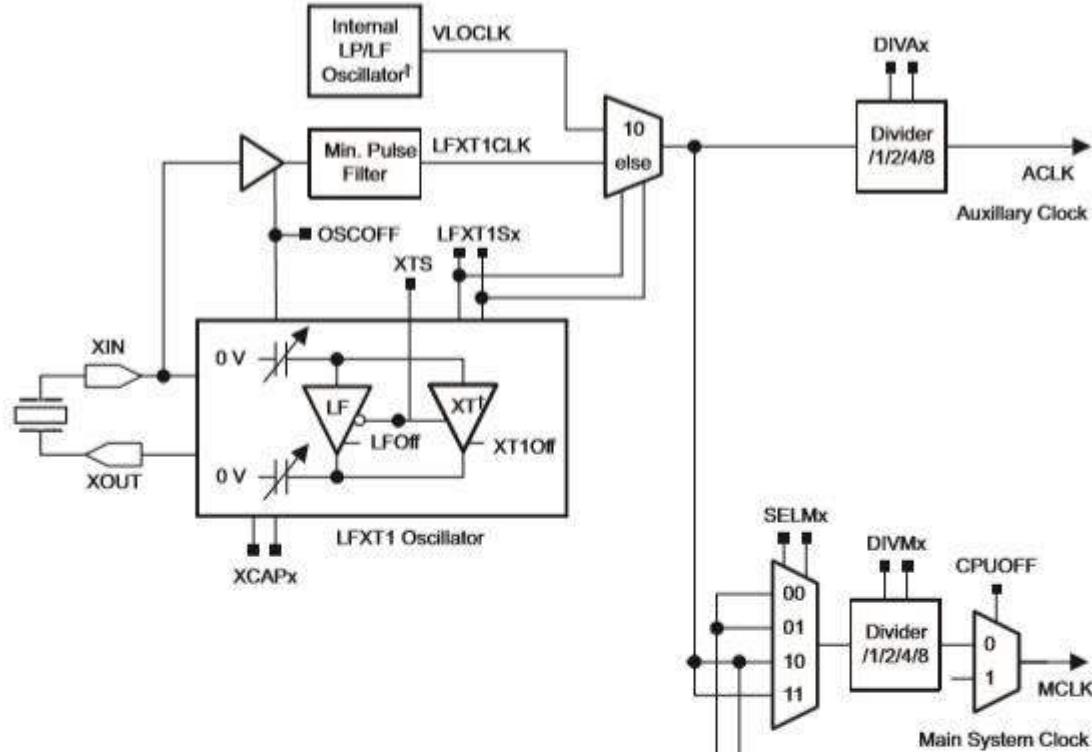
- MSP430 can get clock from 3 sources:
 1. DCO
 2. LFXT1
 3. VLO

Clock Sources

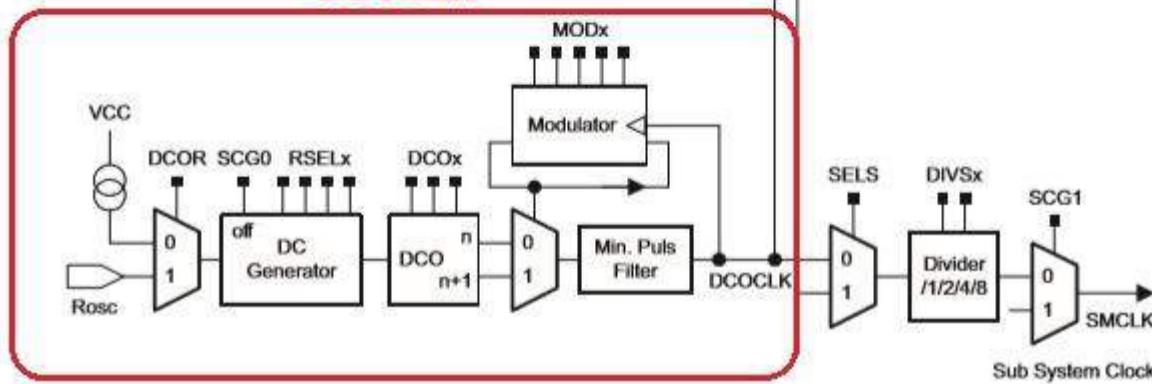
1. DCO

Digitally Controlled Oscillator

- One of the highlights of MSP430, it is basically a highly controllable RC oscillator that starts in less than $1\mu\text{s}$. Therefore starts rapidly at full speed from LPM.
- The DCO frequency can be adjusted by software.
- DCO frequency ranges from 60K to 16MHz.
- **Default frequency => 1.1MHz**



DCOCLK

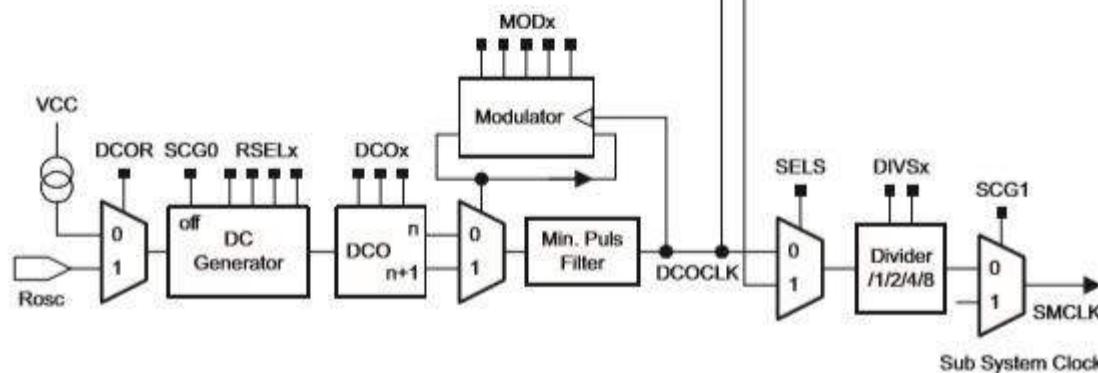
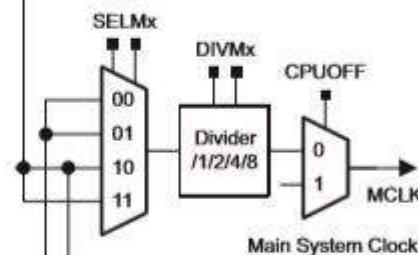
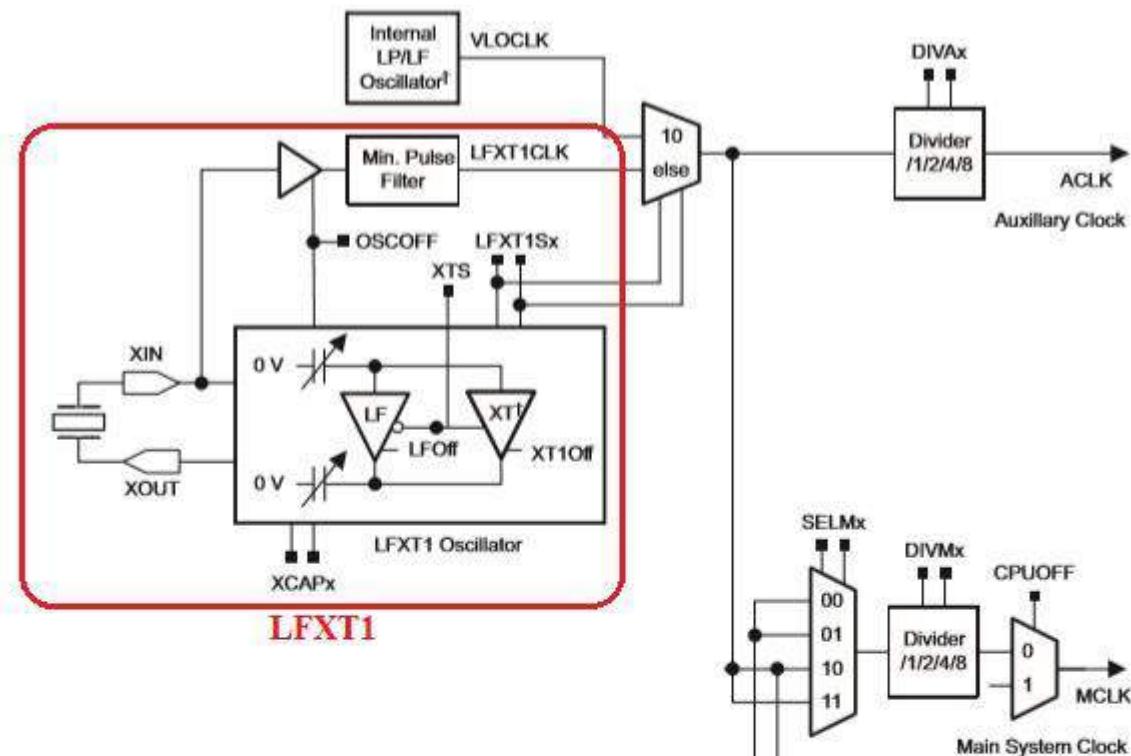


Clock Sources

□ 2. LFXT1



- Low or high frequency crystal oscillator.
- Used with low frequency watch crystal (32 kHz)
- Also used with a high frequency crystal (400 kHz to 16MHz)
(Absent in MSP430G2553)

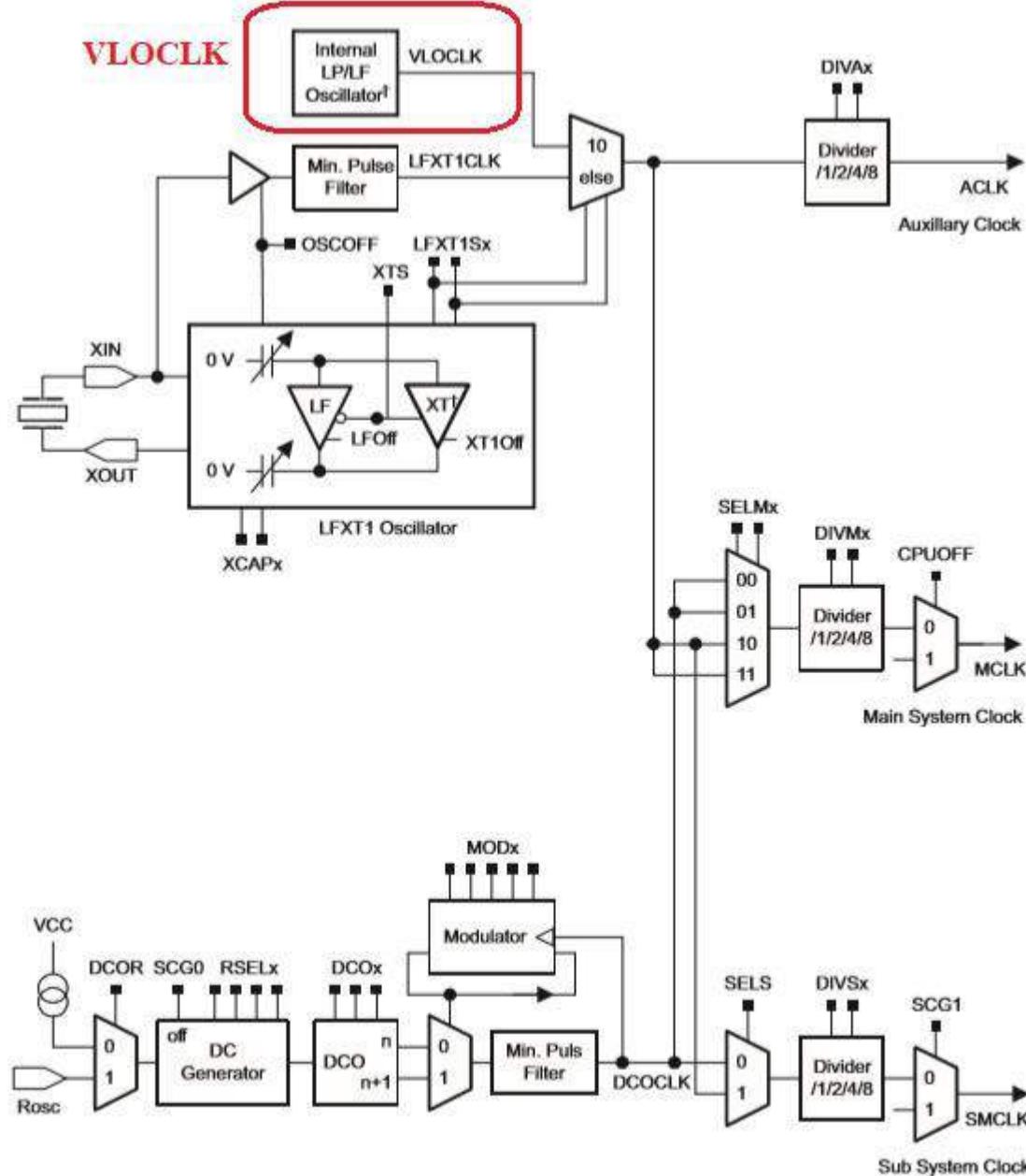


Clock Sources

□ 3. VLO



- Internal very low-power, low-frequency oscillator
- Typical frequency 12kHz



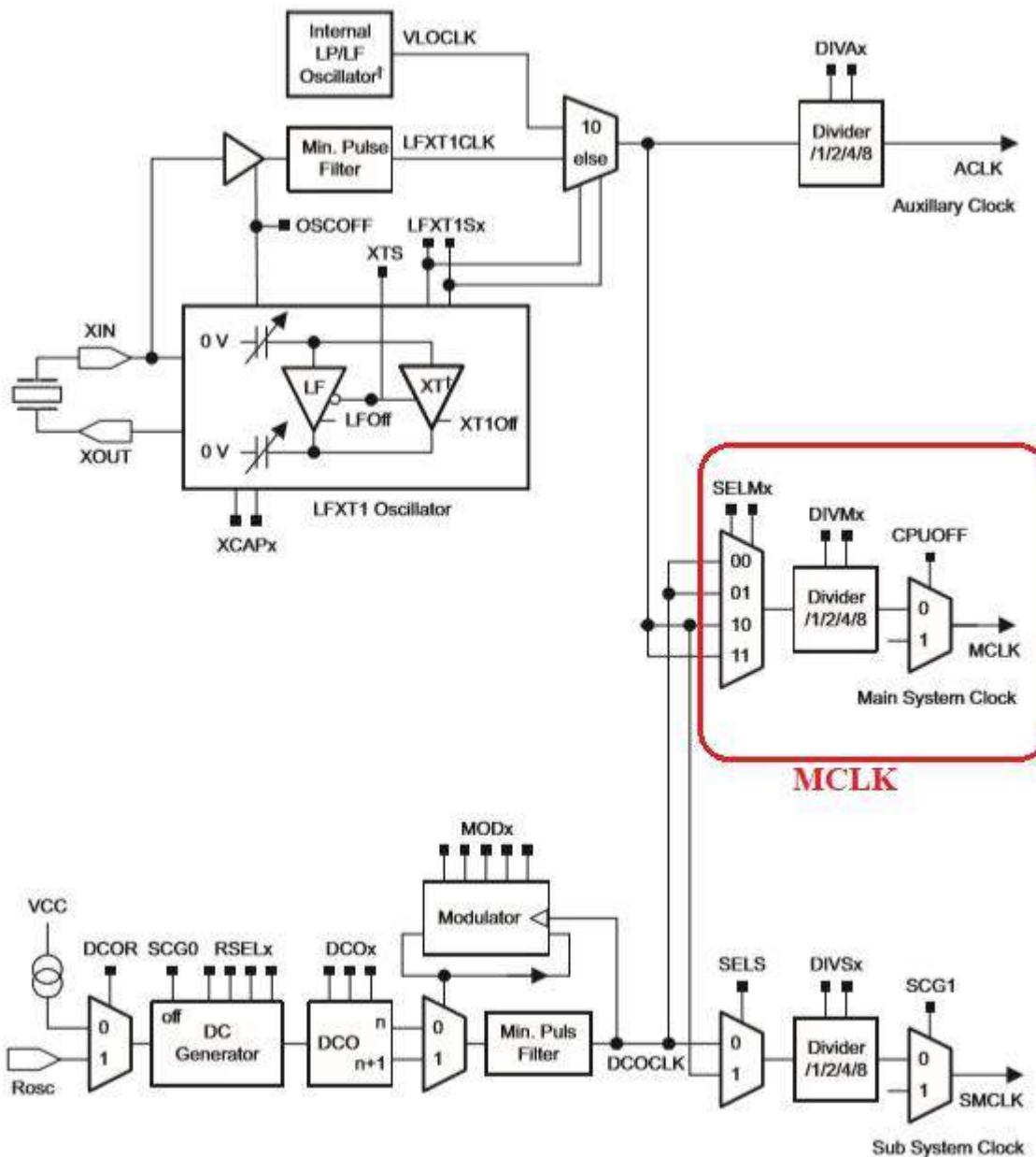
Clock Signals

- Why do we need so many different sources of clock?
Because of conflicting demands of high performance and low power.
We need different kinds of clock for different purposes:
 - Low clock frequency for energy conservation and time keeping
 - High clock frequency for fast reaction to events and fast burst processing capability
 - Clock stability over operating temperature and supply voltage
- Using three internal clock signals, the user can select the best balance of performance and low power consumption.

Clock Signals

□ 1. MCLK: Master clock

- Used by the CPU and a few peripherals.
- Supplied by the DCO with a frequency of around 1.1MHz.
Stabilized by PLL.
- MCLK is software selectable as LFXT1CLK, VLOCLK,
DCOCLK (or XT2CLK, but not present on MSP430G2553).

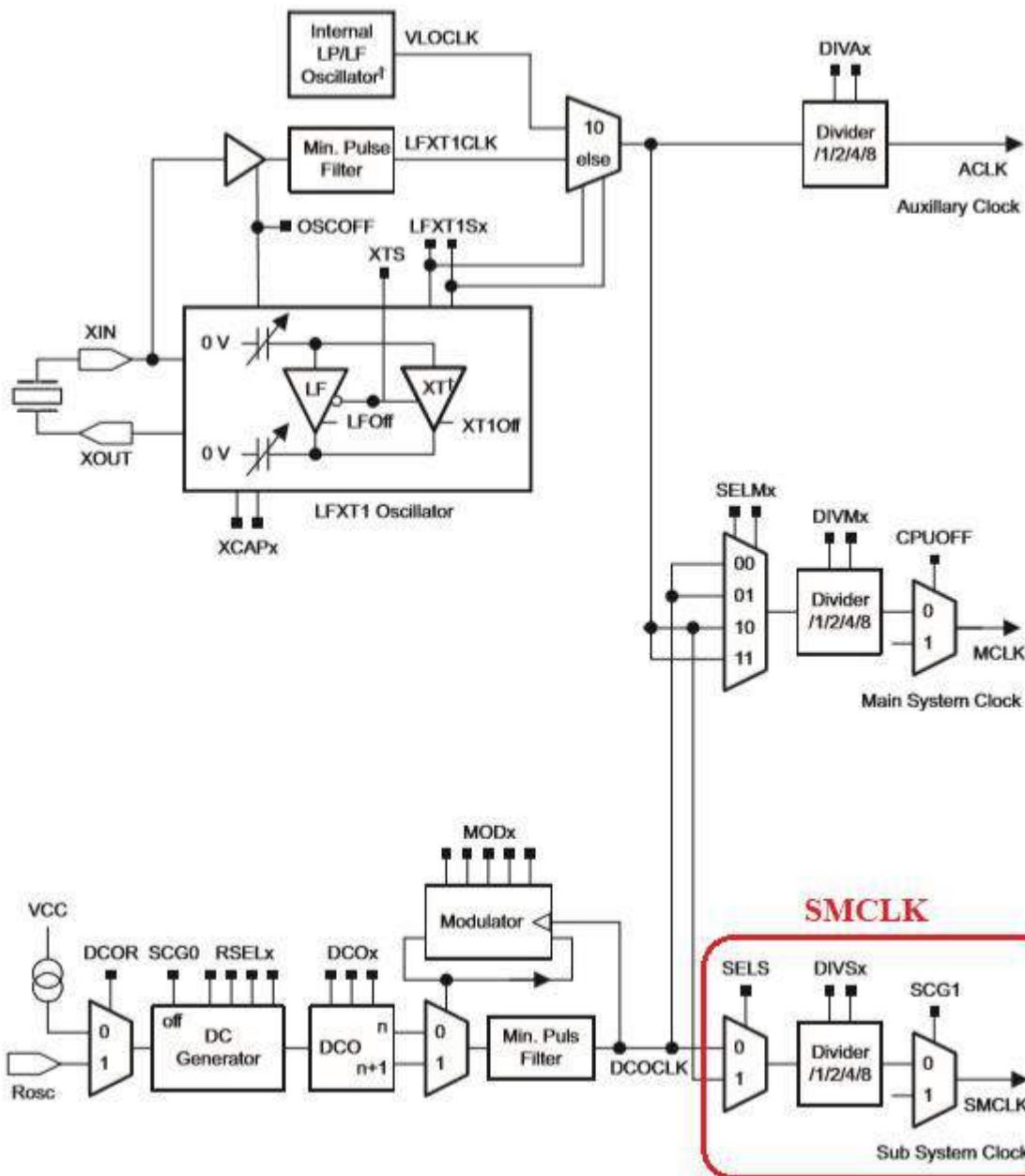


Clock Signals

□ 2. SMCLK: Sub-system Master Clock



- Distributed to peripherals.
- Often same as MCLK.
- Supplied by the DCO with a frequency of around 1.1MHz.
Stabilized by PLL.
- SMCLK is software selectable as LFXT1CLK, VLOCLK,
DCOCLK (or XT2CLK, but not present on MSP430G2553).

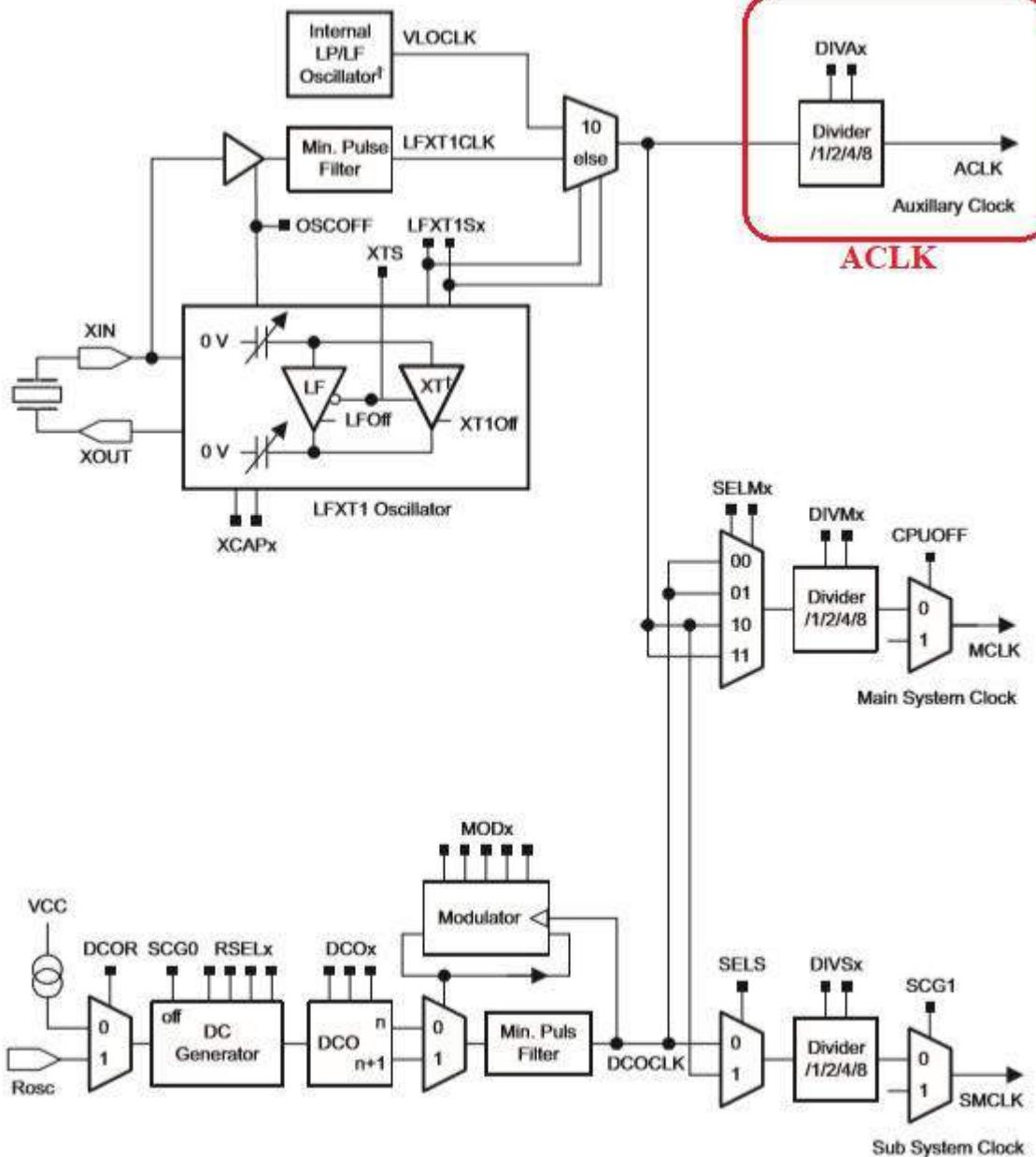


Clock Signals

□ 3. ACLK: Auxiliary Clock



- Distributed among peripherals.
- Sourced from LFXT1CLK or VLOCLK.
- Typically much slower and usually $\leq 32\text{kHz}$.
- ACLK is software selectable as LFXT1CLK or VLOCLK.



Calibrated Frequencies of DCO

□ 4 Calibrated frequencies:

- 1 MHz
- 8 MHz
- 12 MHz
- 16 MHz

□ Sample CCS Statements:

BCSCTL1 = CALBC1_1MHZ; (For range selection)

DCOCTL = CALDCO_1MHZ; (For Freq. selection)

Wish to add crystal?

- Crystals are used when an accurate, stable frequency is needed.
- Crystals are cut from carefully grown, high-quality quartz with specific orientations to give them high stability.
- Traditional crystals oscillate at frequencies of a few MHz.
- Machined into complicated tuning fork shapes to give the low frequency.
- Designed to be most stable at 25 C.
- Crystal connected between XIN & XOUT.

Wish to add crystal?

Disadvantage:

- Frequency is more sensitive to temperature.
- SOLUTION => EXTERNAL OSCILLATOR**
- Requires a **capacitance** to GND from each pin. Value depends on the crystal and is typically 10pF or 22pF.
 - Part of it contributed by stray capacitance from PCB track, therefore kept short.
 - External capacitance needed for many controllers/RTC ICs but integrated in MSP430.

Clock Registers

- **DCOCTL (DCO Control Register)**
- **BCSCTL1 (Basic Clock System Control Register 1)**
- **BCSCTL2 (Basic Clock System Control Register 2):** For MCLK & SMCLK manipulation
- **BCSCTL3 (Basic Clock System Control Register 3):** For external crystal and capacitor selections

DCOCTL: DCO Control Register

	7	6	5	4	3	2	1	0
	DCOx			MODx				
	rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0
DCOx	Bits 7-5 DCO frequency select. These bits select which of the eight discrete DCO frequencies within the range defined by the RSELx setting is selected.							
MODx	Modulator selection. These bits define how often the f_{DCO+1} frequency is used within a period of 32 DCOCLK cycles. During the remaining clock cycles (32-MOD) the f_{DCO} frequency is used. Not useable when DCOx = 7.							

BCSCTL1: Basic Clock Control Register 1

	7	6	5	4	3	2	1	0
	XT2OFF	XTS ⁽¹⁾⁽²⁾	DIVAx			RSELx		
	rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-1	rw-1
XT2OFF	Bit 7	XT2 off. This bit turns off the XT2 oscillator.						
		0 XT2 is on						
		1 XT2 is off if it is not used for MCLK or SMCLK.						
XTS	Bit 6	LFXT1 mode select.						
		0 Low-frequency mode						
		1 High-frequency mode						
DIVAx	Bits 5-4	Divider for ACLK						
		00 /1						
		01 /2						
		10 /4						
		11 /8						
RSELx	Bits 3-0	Range select. Sixteen different frequency ranges are available. The lowest frequency range is selected by setting RSELx = 0. RSEL3 is ignored when DCOR = 1.						

⁽¹⁾ XTS = 1 is not supported in MSP430x20xx and MSP430G2xx devices (see [Figure 5-1](#) and [Figure 5-2](#) for details on supported settings for all devices).

⁽²⁾ This bit is reserved in the MSP430AFE2xx devices.

BCSCTL2: Basic Clock Control Register 2

	7	6	5	4	3	2	1	0
	SEL ^{Mx}		DIV ^{Mx}		SEL ^S		DIV ^{Sx}	DCOR ⁽¹⁾⁽²⁾
SEL ^{Mx}	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
SEL ^{Mx}	Bits 7-6	Select MCLK. These bits select the MCLK source.						
	00	DCOCLK						
	01	DCOCLK						
	10	XT2CLK when XT2 oscillator present on-chip. LFXT1CLK or VLOCLK when XT2 oscillator not present on-chip.						
	11	LFXT1CLK or VLOCLK						
DIV ^{Mx}	Bits 5-4	Divider for MCLK						
	00	/1						
	01	/2						
	10	/4						
	11	/8						
SEL ^S	Bit 3	Select SMCLK. This bit selects the SMCLK source.						
	0	DCOCLK						
	1	XT2CLK when XT2 oscillator present. LFXT1CLK or VLOCLK when XT2 oscillator not present						
DIV ^{Sx}	Bits 2-1	Divider for SMCLK						
	00	/1						
	01	/2						
	10	/4						
	11	/8						
DCOR	Bit 0	DCO resistor select. Not available in all devices. See the device-specific data sheet.						
	0	Internal resistor						
	1	External resistor						

(1) Does not apply to MSP430x20xx or MSP430x21xx devices.
(2) This bit is reserved in the MSP430AFE2xx devices.

BCSCTL3: Basic Clock Control Register 3

	7	6	5	4	3	2	1	0
	XT2Sx		LFXT1Sx ⁽¹⁾		XCAPx ⁽²⁾		XT2OF ⁽³⁾	LFXT1OF ⁽²⁾
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1	r0	r-(1)
XT2Sx	Bits 7-6	XT2 range select. These bits select the frequency range for XT2.						
	00	0.4- to 1-MHz crystal or resonator						
	01	1- to 3-MHz crystal or resonator						
	10	3- to 16-MHz crystal or resonator						
	11	Digital external 0.4- to 16-MHz clock source						
LFXT1Sx	Bits 5-4	Low-frequency clock select and LFXT1 range select. These bits select between LFXT1 and VLO when XTS = 0, and select the frequency range for LFXT1 when XTS = 1.						
	When XTS = 0:							
	00	32768-Hz crystal on LFXT1						
	01	Reserved						
	10	VLOCLK (Reserved in MSP430F21x1 devices)						
	11	Digital external clock source						
	When XTS = 1 (Not applicable for MSP430x20xx devices, MSP430G2xx1/2/3)							
	00	0.4- to 1-MHz crystal or resonator						
	01	1- to 3-MHz crystal or resonator						
	10	3- to 16-MHz crystal or resonator						
	11	Digital external 0.4- to 16-MHz clock source						
	LFXT1Sx definition for MSP430AFE2xx devices:							
	00	Reserved						
	01	Reserved						
	10	VLOCLK						
	11	Reserved						
XCAPx	Bits 3-2	Oscillator capacitor selection. These bits select the effective capacitance seen by the LFXT1 crystal when XTS = 0. If XTS = 1 or if LFXT1Sx = 11 XCAPx should be 00.						
	00	~1 pF						
	01	~6 pF						
	10	~10 pF						
	11	~12.5 pF						
XT2OF	Bit 1	XT2 oscillator fault						
	0	No fault condition present						
	1	Fault condition present						
LFXT1OF	Bit 0	LFXT1 oscillator fault						
	0	No fault condition present						
	1	Fault condition present						

⁽¹⁾ MSP430G22x0: The LFXT1Sx bits should be programmed to 10b during the initialization and start-up code to select VLOCLK (for more details refer to Digital I/O chapter). The other bits are reserved and should not be altered.
⁽²⁾ This bit is reserved in the MSP430AFE2xx devices.
⁽³⁾ Does not apply to MSP430x2xx, MSP430x21xx, or MSP430x22xx devices.

Setting the frequency of the DCO

The frequency of DCOCLK is set by the following functions:

- The four RSELx bits select one of sixteen nominal frequency ranges for the DCO. These ranges are defined in the datasheet.
- The three DCOx bits divide the DCO range selected by the RSELx bits into 8 frequency steps, separated by approximately 10%.
- The five MODx bits, switch between the frequency selected by the DCOx bits and the next higher frequency set by DCOx+1. When DCOx = 07h, the MODx bits have no effect because the DCO is already at the highest setting for the selected RSELx range.

Setting the frequency of the DCO



MSP430G2x53
MSP430G2x13

www.ti.com

SLAS735J-APRIL 2011-REVISED MAY 2013

DCO Frequency

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER	TEST CONDITIONS	V _{cc}	MIN	TYP	MAX	UNIT
V _{cc} Supply voltage	RSELx < 14		1.8	3.6	3.6	V
	RSELx = 14		2.2	3.6		
	RSELx = 15		3	3.6		
f _{DCO(0,0)}	DCO frequency (0, 0) RSELx = 0, DCOx = 0, MODx = 0	3 V	0.06	0.14	0.14	MHz
f _{DCO(0,3)}	DCO frequency (0, 3) RSELx = 0, DCOx = 3, MODx = 0	3 V	0.07	0.17	0.17	MHz
f _{DCO(1,3)}	DCO frequency (1, 3) RSELx = 1, DCOx = 3, MODx = 0	3 V		0.15	0.15	MHz
f _{DCO(2,3)}	DCO frequency (2, 3) RSELx = 2, DCOx = 3, MODx = 0	3 V		0.21	0.21	MHz
f _{DCO(3,3)}	DCO frequency (3, 3) RSELx = 3, DCOx = 3, MODx = 0	3 V		0.30	0.30	MHz
f _{DCO(4,3)}	DCO frequency (4, 3) RSELx = 4, DCOx = 3, MODx = 0	3 V		0.41	0.41	MHz
f _{DCO(5,3)}	DCO frequency (5, 3) RSELx = 5, DCOx = 3, MODx = 0	3 V		0.58	0.58	MHz
f _{DCO(6,3)}	DCO frequency (6, 3) RSELx = 6, DCOx = 3, MODx = 0	3 V	0.54	1.06	1.06	MHz
f _{DCO(7,3)}	DCO frequency (7, 3) RSELx = 7, DCOx = 3, MODx = 0	3 V	0.80	1.50	1.50	MHz
f _{DCO(8,3)}	DCO frequency (8, 3) RSELx = 8, DCOx = 3, MODx = 0	3 V		1.6	1.6	MHz
f _{DCO(9,3)}	DCO frequency (9, 3) RSELx = 9, DCOx = 3, MODx = 0	3 V		2.3	2.3	MHz
f _{DCO(10,3)}	DCO frequency (10, 3) RSELx = 10, DCOx = 3, MODx = 0	3 V		3.4	3.4	MHz
f _{DCO(11,3)}	DCO frequency (11, 3) RSELx = 11, DCOx = 3, MODx = 0	3 V		4.25	4.25	MHz
f _{DCO(12,3)}	DCO frequency (12, 3) RSELx = 12, DCOx = 3, MODx = 0	3 V	4.30	7.30	7.30	MHz
f _{DCO(13,3)}	DCO frequency (13, 3) RSELx = 13, DCOx = 3, MODx = 0	3 V	6.00	7.8	9.60	MHz
f _{DCO(14,3)}	DCO frequency (14, 3) RSELx = 14, DCOx = 3, MODx = 0	3 V	8.80		13.8	MHz
f _{DCO(15,3)}	DCO frequency (15, 3) RSELx = 15, DCOx = 3, MODx = 0	3 V	12.0		18.5	MHz
f _{DCO(15,7)}	DCO frequency (15, 7) RSELx = 15, DCOx = 7, MODx = 0	3 V	16.0		28.0	MHz
S _{RSEL}	Frequency step between range RSEL and RSEL+1	S _{RSEL} = f _{DCO(RSEL+1,DCO)} /f _{DCO(RSEL,DCO)}	3 V		1.35	ratio
S _{DCO}	Frequency step between tap DCO and DCO+1	S _{DCO} = f _{DCO(RSEL,DCO+1)} /f _{DCO(RSEL,DCO)}	3 V		1.08	ratio
Duty cycle	Measured at SMCLK output	3 V		50	%	

Clock Code Example

```
1 #include <msp430.h>
2
3 #define LED BIT7
4
5 #define SW1 BIT3           // 1.5 kHz
6 #define SW2 BIT4           // 3 kHz
7 #define SW3 BIT5           // 12 kHz
8
9 volatile unsigned int i;          // Volatile to prevent removal
10
11 /**
12 * @brief
13 * Function to take input from 3 switches and change CPU clock accordingly
14 */
15 void switch_input()
16 {
17
18     if(!(P1IN & SW1))          // If SW1 is Pressed
19     {
20         __delay_cycles(2000);    // Wait 20ms to debounce
21         while(!(P1IN & SW1));   // Wait till SW Released
22         __delay_cycles(2000);    // Wait 20ms to debounce
23
24         BCSCTL2 &=~ (BIT5 + BIT4); //Reset VLO divider
25         BCSCTL2 |= (BIT5 + BIT4); //VLO = 12kHz/8 = 1.5kHz
26     }
27
28
29     if(!(P1IN & SW2))          // If SW is Pressed
30     {
31         __delay_cycles(2000);    // Wait 20ms to debounce
32         while(!(P1IN & SW2));   // Wait till SW Released
33         __delay_cycles(2000);    // Wait 20ms to debounce
34
35         BCSCTL2 &=~ (BIT5 + BIT4); //Reset VLO divider
36         BCSCTL2 |= (BIT4);        //VLO = 12kHz/4 = 3kHz
37     }
38 }
```

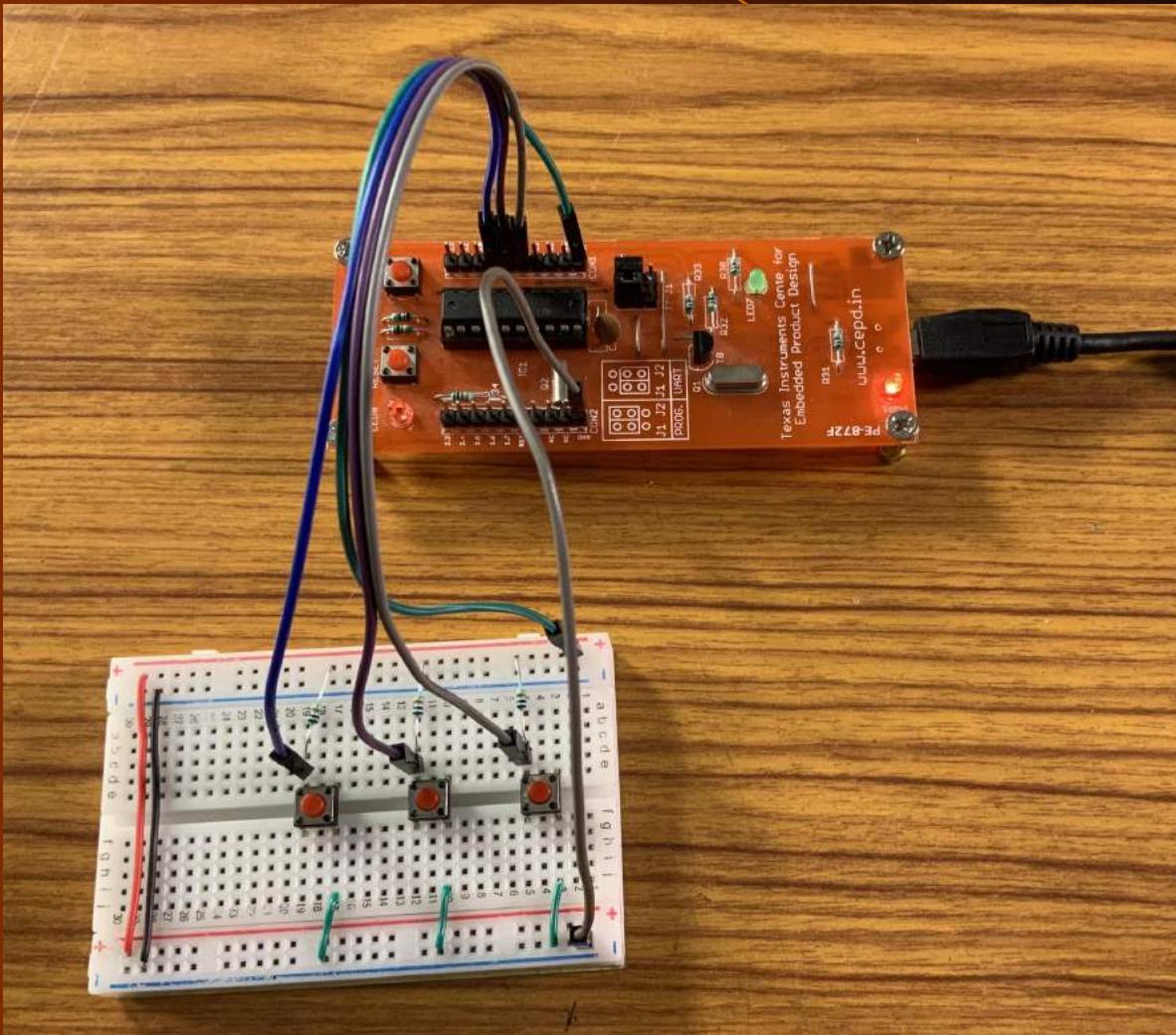
Clock Code Example

```
39     if(!(P1IN & SW3))           // If SW is Pressed
40     {
41
42         __delay_cycles(2000);    // Wait 20ms to debounce
43         while(!(P1IN & SW3));   // Wait till SW Released
44         __delay_cycles(2000);    // Wait 20ms to debounce
45
46         BCSCTL2 &=~ (BIT5 + BIT4);      //VLO = 12kHz/1 = 12kHz
47     }
48
49 }
50
51 /**
52 * @brief
53 * These settings are wrt enabling GPIO on Lunchbox
54 */
55 void register_settings_for_GPIO()
56 {
57     P1DIR |= LED;                  //P1.7 is set as Output
58     P1DIR &=~ (SW1 + SW2 + SW3);   //P1.3, P1.4, P1.5 are set as Input
59 }
60
61
62 /**
63 * @brief
64 * These settings are w.r.t enabling VLO on Lunch Box
65 */
66 void register_settings_for_VLO()
67 {
68     BCSCTL3 |= LFXT1S_2;          // LFXT1 = VLO
69     do{
70         IFG1 &= ~OFIFG;            // Clear oscillator fault flag
71         for (i = 50000; i; i--);   // Delay
72     } while (IFG1 & OFIFG);       // Test osc fault flag
73
74     BCSCTL2 |= SELM_3;            // MCLK = VLO
75 }
76
```

Clock Code Example

```
76
77 /**
78 * main.c
79 */
80 int main(void)
81 {
82     WDTCTL = WDTPW | WDTHOLD;           //Stop watchdog timer
83
84     register_settings_for_VLO();        //Register settings for VLO
85     register_settings_for_GPIO();       //Register settings for GPIO
86
87
88     while(1)
89     {
90         switch_input();                //Switch Input
91
92         P1OUT ^= LED;                //LED Toggle
93         for (i = 100; i > 0; i--);
94
95     }
96 }
```

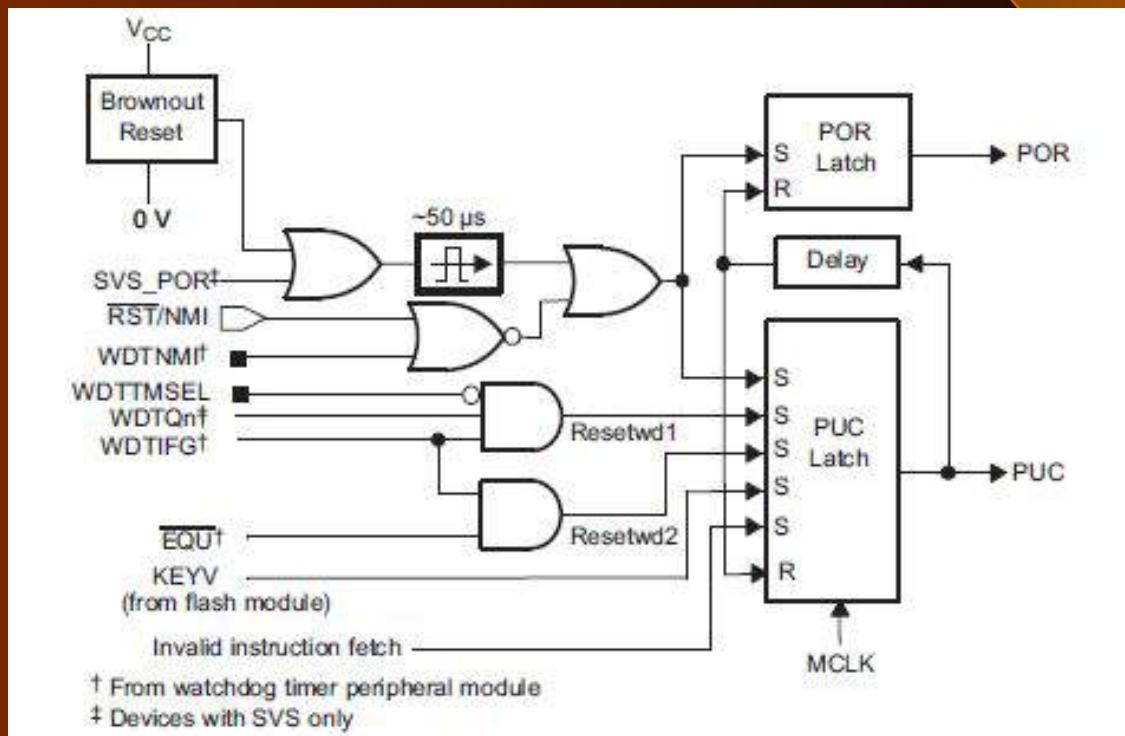
Circuit For The Clock Source Example Code



MSP430 Resets

There are two types of resets:

1. Power on reset (POR)
2. Power up clear (PUC)



Power On Reset

A POR is a device reset and is generated by conditions related to hardware. It occurs in any of the following conditions:

1. Powering up the device
2. Brownout Reset: A POR is raised even when the supply voltage drops to so low a value that the device may not work correctly.
3. A high to low signal on the RST/NMI pin when configured in the reset mode. (User Reset)

Power Up Clear

A PUC is generated by software conditions. It is also always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

1. A POR signal
2. Watchdog timer expiration when in watchdog mode only
3. Watchdog timer security key violation (An attempt is made to write to the watchdog control register WDTCTL without the correct password 0x05A)
4. A Flash memory security key violation
5. A CPU instruction fetch from the peripheral address range of 0000h to 01FFh

Device Initial Conditions After System Reset

After a POR, the initial MSP430 conditions are:

- The RST/NMI pin is configured in the reset mode.
- I/O pins are switched to input mode.
- Other peripheral modules and registers are initialized as described in the user guide.

The value is shown under each bit in the description of the registers. For example, rw–0 means that a bit can be both read and written and is initialized to 0 after a PUC. Whereas, rw–(0) shows that a bit is initialized to 0 only after a POR; it retains its value through a PUC.

Device Initial Conditions After System Reset

- Status register (SR) is reset. This means that the device operates at full power, even though it might have been in a low-power mode before the reset occurred.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with address contained at reset vector location (FFFEh).

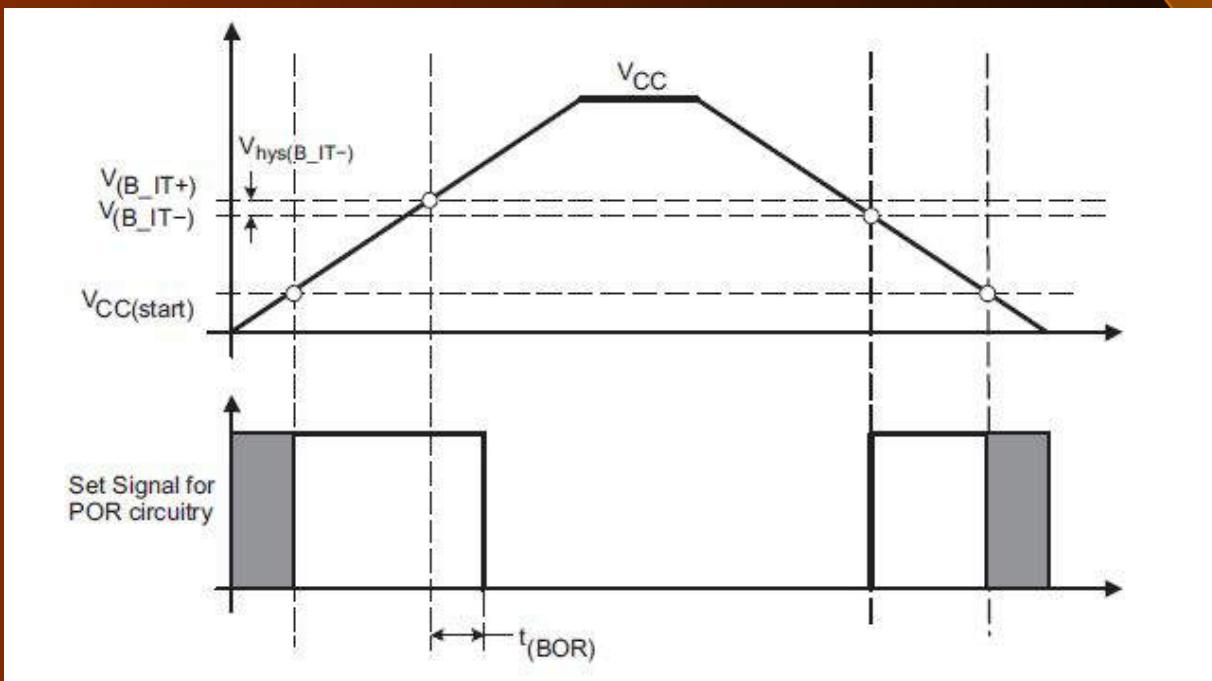
Software Initialisations after System Reset

After a system reset, the software must:

- Initialize the watchdog timer, usually to turn it off
- Configure peripheral modules

Brownout Reset

- The brownout reset circuit detects low supply voltages such as when a supply voltage is applied to or removed from the VCC terminal.
- The brownout reset circuit resets the device by triggering a POR signal when power is applied or removed.



Brownout Reset

- The POR signal becomes active when VCC crosses the VCC(start) level.
- It remains active until VCC crosses the $V(B_{IT+})$ threshold and the delay $t(BOR)$ elapses.
- The delay $t(BOR)$ is adaptive being longer for a slow ramping VCC. The hysteresis $V_{hys}(B_{IT-})$ ensures that the supply voltage must drop below $V(B_{IT-})$ to generate another POR signal from the brownout reset circuitry.

Watchdog Timer

- The primary function of the watchdog timer (WDT+) module is to perform a controlled system restart after a software problem occurs.
- If the selected time interval expires, a system reset is generated.
- If the watchdog function is not needed in an application, the module can be disabled or configured as an interval timer and can generate interrupts at selected time intervals.

Watchdog Timer Operating Modes

The watchdog timer+ module has two operating modes:

1. Watchdog mode:
 - After a PUC, the WDT+ module is configured in the watchdog mode with an initial 32768 cycle reset interval using the DCOCLK.
 - The user must setup, halt, or clear the WDT+ prior to the expiration of the initial reset interval or another PUC will be generated.
 - When the WDT+ is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC.

Watchdog Timer Operating Modes

2. Interval mode

- This mode can be used to provide periodic interrupts.
- In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval.
- A PUC is not generated in interval timer mode at expiration of the selected timer interval and the WDTIFG enable bit WDTIE remains unchanged.
- When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode.

Watchdog Timer Registers

- WDTCTL: Watchdog Timer+ Control Register:
- IE1: SFR Interrupt Enable Register 1
- IFG1: SFR Interrupt Flag Register 1

WDTCTL Register

- WDTCTL is a 16-bit, password-protected, read/write register.
- Any read or write access must use word instructions and write accesses must include the write password 05Ah in the upper byte.
- Any write to WDTCTL with any value other than 05Ah in the upper byte is a security key violation and triggers a PUC system reset regardless of timer mode.
- Any read of WDTCTL reads 069h in the upper byte.

WDTCTL Register

15	14	13	12	11	10	9	8
WDTPW. Read as 069h Must be written as 05Ah							
7	6	5	4	3	2	1	0
WDTHOLD	WDTNMIES	WDTNMI	WDTTMSEL	WDTCNTCL	WDTSSSEL	WDTISx	
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-0	rw-0	rw-0
WDTPW	Bits 15-8	Watchdog timer+ password. Always read as 069h. Must be written as 05Ah, or a PUC is generated.					
WDTHOLD	Bit 7	Watchdog timer+ hold. This bit stops the watchdog timer+. Setting WDTHOLD = 1 when the WDT+ is not in use conserves power.					
	0	Watchdog timer+ is not stopped					
	1	Watchdog timer+ is stopped					
WDTNMIES	Bit 6	Watchdog timer+ NMI edge select. This bit selects the interrupt edge for the NMI interrupt when WDTNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when WDTIE = 0 to avoid triggering an accidental NMI.					
	0	NMI on rising edge					
	1	NMI on falling edge					
WDTNMI	Bit 5	Watchdog timer+ NMI select. This bit selects the function for the RST/NMI pin.					
	0	Reset function					
	1	NMI function					
WDTTMSEL	Bit 4	Watchdog timer+ mode select					
	0	Watchdog mode					
	1	Interval timer mode					
WDTCNTCL	Bit 3	Watchdog timer+ counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.					
	0	No action					
	1	WDTCNT = 0000h					
WDTSSSEL	Bit 2	Watchdog timer+ clock source select					
	0	SMCLK					
	1	ACLK					
WDTISx	Bits 1-0	Watchdog timer+ interval select. These bits select the watchdog timer+ interval to set the WDTIFG flag and/or generate a PUC.					
	00	Watchdog clock source /32768					
	01	Watchdog clock source /8192					
	10	Watchdog clock source /512					
	11	Watchdog clock source /64					

IE1: SFR Interrupt Enable Register 1

- WDTIE bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode.

IFG1: SFR Interrupt Flag Register 1

Address	7	6	5	4	3	2	1	0
02h				NMIIFG	RSTIFG	PORIFG	OFIFG	WDTIFG
				rw-0	rw-(0)	rw-(1)	rw-1	rw-(0)
WDTIFG					Set on watchdog timer overflow (in watchdog mode) or security key violation. Reset on V _{CC} power-on or a reset condition at the RST/NMI pin in reset mode.			
OFIFG					Flag set on oscillator fault.			
PORIFG					Power-On Reset interrupt flag. Set on V _{CC} power-up.			
RSTIFG					External reset interrupt flag. Set on a reset condition at RST/NMI pin in reset mode. Reset on V _{CC} power-up.			
NMIIFG					Set via RST/NMI pin			

How to stop the watchdog timer?

The watchdog timer is stopped by the following statement:

`WDTCTL = WDTPW + WDTHOLD;`

This instruction sets the password (WDTPW) as 5Ah and the bit to stop the timer (WDTHOLD).

Determining the Reset Source: Code Example

```
1 #include <msp430.h>
2
3 #define LED1 BIT0                                // main() has started
4 #define LED2 BIT1                                // program is executing
5 #define LEDA BIT2                                // POR
6 #define LEDB BIT3                                // WDT
7 #define LEDC BIT4                                // RST
8
9 /**
10 * @brief
11 * These settings are wrt enabling GPIO on Lunchbox
12 */
13 void register_settings_for_GPIO()
14{
15    P1DIR |= (LED1 + LED2 + LEDA + LEDB + LEDC);      // P1.0 to P1.4 as output
16
17    P1OUT |= LED1;                                  // LED1 representing main() is set
18
19    P1OUT &=~ (LED2 + LEDA + LEDB + LEDC);        // Turning all other LEDs OFF
20}
21
22 /**
23 * @brief
24 * These settings are w.r.t check source of reset on Lunch Box
25 */
26 void checking_reset_source()
27{
28    if (IFG1 & PORIFG)                            // Check for Power on Reset flag (POR)
29    {
30        P1OUT |= LEDA;                            // Turning LEDA On
31        P1OUT &= ~LEDB;                          // Turning LEDB Off
32        P1OUT &= ~LEDC;                          // Turning LEDC Off
33
34        /* Settings for Watchdog Timer register
35           Giving Watchdog Timer Password
36           Clearing Watchdog counter to 0000h
37           Watchdog Source select --> ACLK
38           Watchdog Timer interval set to generate watchdog reset at 2 secs
39        */
40        WDTCTL = (WDTPW + WDTCNTCL + WDTSEL + WDTIS0);
41
42        IFG1 &=~ PORIFG;                         // Clearing Power on Reset flag
43    }
44}
```

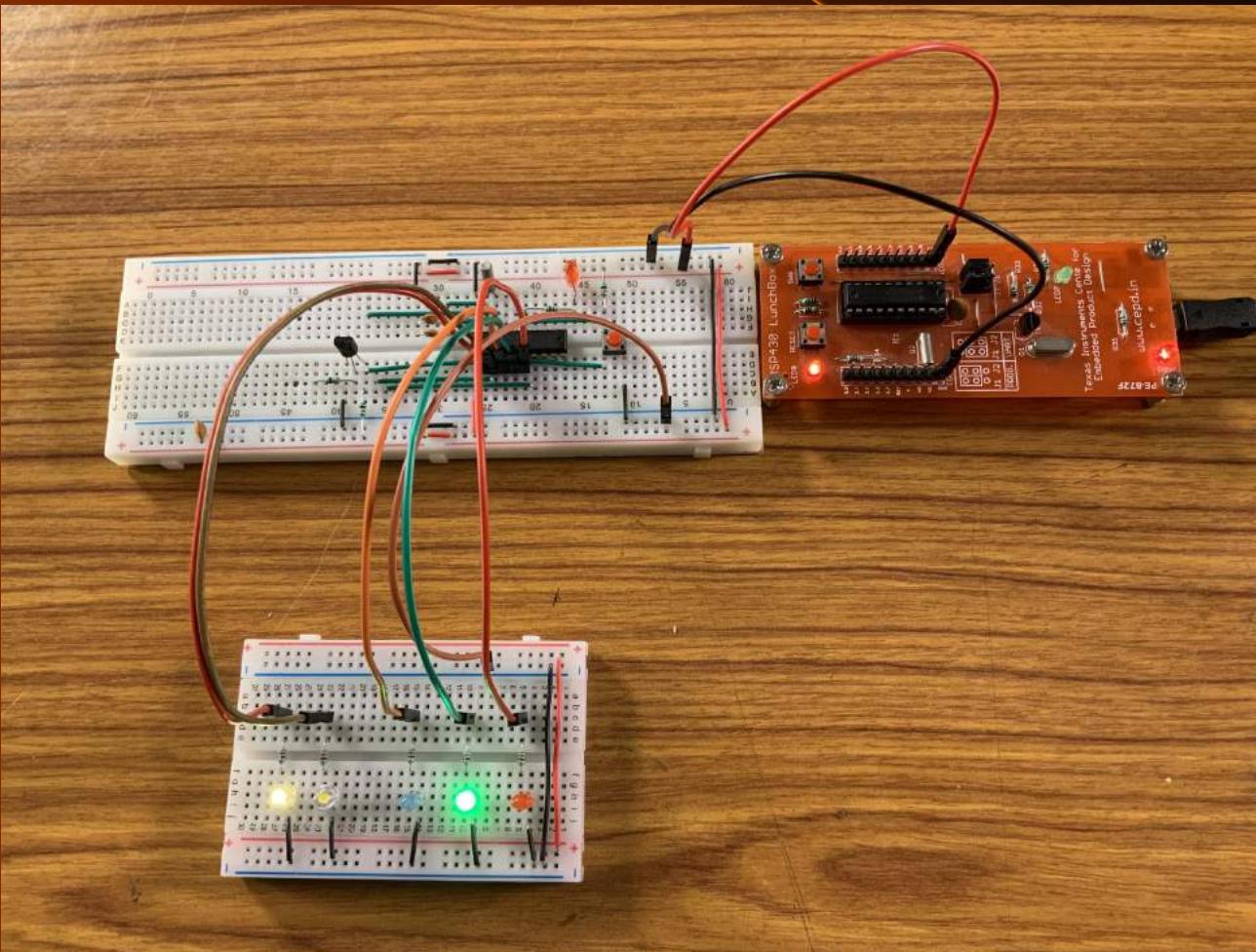
Determining the Reset Source: Code Example

```
43     }
44
45     else if (IFG1 & RSTIFG)           // Check for External Reset flag (RST)
46     {
47         P1OUT |= LEDC;             // Turning LEDC On
48         P1OUT &= ~LEDA;            // Turning LEDA Off
49         P1OUT &= ~LEDB;            // Turning LEDB Off
50
51         /* Settings Watchdog Timer register
52             Giving Watchdog Timer Password
53             Clearing Watchdog counter to 0000h
54             Watchdog Source select --> ACLK
55             Watchdog Timer interval set to generate watchdog reset at 2 secs
56         */
57         WDTCTL = (WDTPW + WDTCNTCL + WDTSEL + WDTIS0);
58
59         IFG1 &=~ RSTIFG;          // Clearing External Reset flag
60     }
61
62     else if (IFG1 & WDTIFG)        // Check for Watch Dog Reset flag (WDT)
63     {
64         P1OUT |= LEDB;             // Turning LEDB On
65         P1OUT &= ~LEDA;            // Turning LEDA Off
66         P1OUT &= ~LEDC;            // Turning LEDC Off
67
68         IFG1 &=~ PORIFG;          // Clearing Power on Reset flag
69         IFG1 &=~ RSTIFG;          // Clearing External Reset flag
70     }
71
72 }
```

Determining the Reset Source: Code Example

```
73
74 /*@brief entry point for the code*/
75 void main(void)
76 {
77     WDTCTL = WDTPW | WDTHOLD;    // stop watch dog timer
78     volatile unsigned int i;
79
80     BCSCTL1 |= DIVA_3;          // Dividing ACLK by 8, 32768Hz/8 = 4096Hz
81
82     register_settings_for_GPIO();
83
84     /* This loop checks for Oscillator fault flag to reset means
85      it delays execution until external crystal is Power On */
86
87     do
88     {
89         IFG1 &= ~OFIFG;           // Clear oscillator fault flag
90         for (i = 10000; i ; i--); // Delay, minimum value of i = 5000
91     } while (IFG1 & OFIFG);    // Test osc fault flag
92
93     checking_reset_source();
94
95     while(1)
96     {
97         P1OUT ^= LED2;          // Toggle LED
98         __delay_cycles(1000000);
99     }
100 }
```

Circuit For The Reset Source Example Code





Thank you!

Introduction to Embedded System Design

Interrupts in MSP430

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Why Interrupts?

- Polling external events to fulfill computational requirements.
- Multiple inputs require round robin querying method
- Good for small number of inputs.

Format-I (Double Operand) Instruction Cycles and Lengths

Addressing Mode		No. of Cycles	Length of Instruction	Example	
Src	Dst				
Rn	Rm	1	1	MOV	R5, R8
	PC	2	1	BR	R9
	x(Rm)	4	2	ADD	R5, 4 (R6)
	EDE	4	2	XOR	R8, EDE
	&EDE	4	2	MOV	R5, &EDE
@Rn	Rm	2	1	AND	@R4, R5
	PC	2	1	BR	@R8
	x(Rm)	5	2	XOR	@R5, 8 (R6)
	EDE	5	2	MOV	@R5, EDE
	&EDE	5	2	XOR	@R5, &EDE
@Rn+	Rm	2	1	ADD	@R5+, R6
	PC	3	1	BR	@R9+
	x(Rm)	5	2	XOR	@R5, 8 (R6)
	EDE	5	2	MOV	@R9+, EDE
	&EDE	5	2	MOV	@R9+, &EDE
#N	Rm	2	2	MOV	#20, R9
	PC	3	2	BR	#2AEh
	x(Rm)	5	3	MOV	#0300h, 0 (SP)
	EDE	5	3	ADD	#33, EDE
	&EDE	5	3	ADD	#33, &EDE

Format-I (Double Operand) Instruction Cycles and Lengths

Addressing Mode		No. of Cycles	Length of Instruction	Example	
Src	Dst				
x(Rn)	Rm	3	2	MOV	2 (R5), R7
	PC	3	2	BR	2 (R6)
	TONI	6	3	MOV	4 (R7), TONI
	x(Rm)	6	3	ADD	4 (R4), 6 (R9)
	&TONI	6	3	MOV	2 (R4), &TONI
EDE	Rm	3	2	AND	EDE, R6
	PC	3	2	BR	EDE
	TONI	6	3	CMP	EDE, TONI
	x(Rm)	6	3	MOV	EDE, 0 (SP)
	&TONI	6	3	MOV	EDE, &TONI
&EDE	Rm	3	2	MOV	&EDE, R8
	PC	3	2	BRA	&EDE
	TONI	6	3	MOV	&EDE, TONI
	x(Rm)	6	3	MOV	&EDE, 0 (SP)
	&TONI	6	3	MOV	&EDE, &TONI

Introduction to Interrupts

An interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request or event.

Where are Interrupts Used?

- Urgent tasks that must be executed promptly at higher priority than main code as well as concurrently with the main code.
- Infrequent tasks, such as handling slow input from humans.
This saves overhead of polling.
 - For example, if you continuously poll (read) an input to wait for a switch to be pressed, your CPU Load is very high. On the other hand, in case the switch is connected to a pin which is configured for an interrupt, your CPU is free to do other tasks or maybe even go to sleep!
- Waking the CPU from sleep. Particularly important for MSP430 which typically spends much of its time in LPM and can be awakened only by interrupt.

Interrupt Versus Function/Subroutine

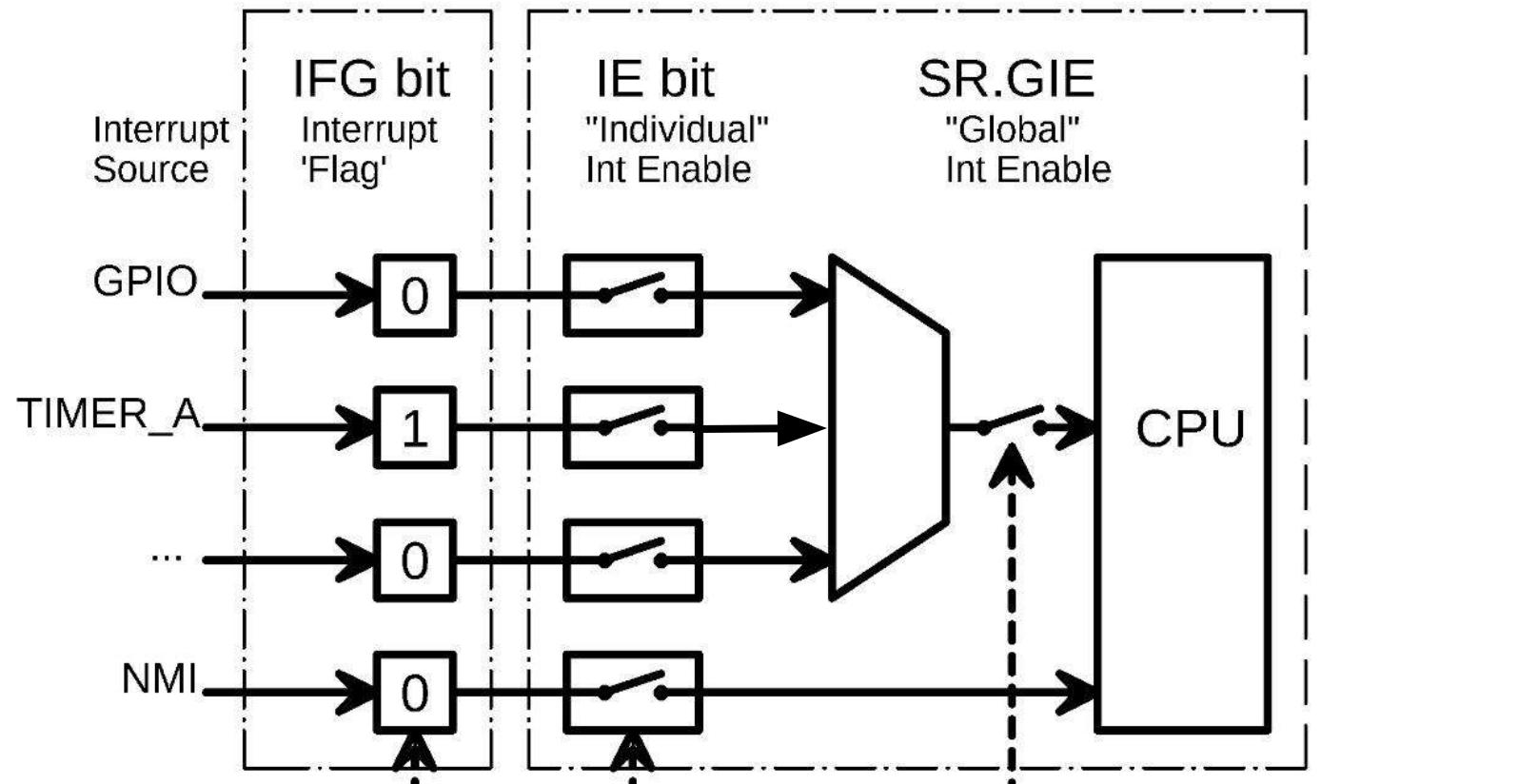
- An interrupt is initiated by an external/internal signal rather than from execution of an instruction (function call).
- An interrupt procedure usually stores all information necessary to define the state of CPU rather than storing only the program counter.
- The address of the interrupt service program is determined by hardware.

Interrupts

- The code which handles an interrupt is called an Interrupt Handler or Interrupt Service Routine (ISR).
- Interrupts can be requested by most peripheral modules like Timers, GPIOs and some in the core of the MCU, such as clock generators.
- Each interrupt has a flag, which is raised (set) when condition of interrupt occurs.
- There are three types of interrupts in MSP430:
 - System Reset
 - Non Maskable Interrupts
 - Maskable Interrupts

Maskable vs Non-Maskable Interrupts

- Most interrupts are maskable (i.e. they can be suspended).
- They are effective only if general interrupt enable (GIE) is set in Status Register (SR).
- Non-maskable interrupts cannot be suppressed by GIE, but are enabled by individual interrupt enable bits.



NMI

MSP430 supports a Non-Maskable Interrupt (NMI). This interrupt is not disabled by GIE bit. It's mainly used for critical external system events.

Global Interrupt Enable(GIE)

Enables all maskable interrupts
 Enable: `_bis_SR_register(GIE);`
 Disable: `_bic_SR_register(GIE);`

Non Maskable Interrupts

- A Non-maskable NMI interrupt can be generated by three sources:
 - Oscillator Fault, OFIFG.
 - Access violation to memory.
 - NMI (non maskable interrupt) pin if it has been configured for interrupt rather than reset.

RST/NMI Pin

- At power-up, the RST/NMI pin is configured as the reset pin.
- The function of the RST/NMI pins is selected in the watchdog control register WDTCTL.
- If the RST/NMI pin is set to the reset function, the CPU is held in the reset state as long as the RST/NMI pin is held low. After the input changes to a high state, the CPU starts program execution at the word address stored in the reset vector, 0FFEh, and the RSTIFG flag is set.
- If the RST/NMI pin is configured by user software as NMI, a signal edge selected by the WDTNMIES bit generates an NMI interrupt if the NMIIIE bit is set. The RST/NMI flag NMIIIFG is also set.

Vectored Interrupts

- MSP430 uses vectored interrupt.
- In a vectored interrupt, the source that interrupts supplies the branch information to the computer.
- Vector address is stored in the vector table.
- Each interrupt vector has a distinct priority, which is used to select which vector is taken if more than 1 interrupt is active.
- Priority is fixed: cannot be changed by user. A higher address means higher priority.
- Reset vector has highest address 0xFFFF.

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range ⁽¹⁾	PORIFG RSTIFG WDTIFG KEYV ⁽²⁾	Reset	0FFF Eh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG ⁽²⁾⁽³⁾	(non)-maskable (non)-maskable (non)-maskable	0FFF Ch	30
Timer1_A3	TA1CCR0 CCIFG ⁽⁴⁾	maskable	0FFF Ah	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG ⁽²⁾⁽⁴⁾	maskable	0FFF 8h	28
Comparator_A+	CAIFG ⁽⁴⁾	maskable	0FFF 6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF 4h	26
Timer0_A3	TA0CCR0 CCIFG ⁽⁴⁾	maskable	0FFF 2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG ⁽⁵⁾⁽⁴⁾	maskable	0FFF 0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG ⁽²⁾⁽⁵⁾	maskable	0FFE Eh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG ⁽²⁾⁽⁶⁾	maskable	0FFE Ch	22
ADC10 (MSP430G2x53 only)	ADC10IFG ⁽⁴⁾	maskable	0FFE Ah	21
			0FFE 8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE 6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE 4h	18
			0FFE 2h	17
			0FFE 0h	16
See ⁽⁷⁾			0FFD Eh	15
See ⁽⁸⁾			0FFD Eh to 0FFC 0h	14 to 0, lowest

Interrupt Flag

- The interrupt flag is cleared automatically for vectors that have a single source.
- Flags remain set for servicing by software if the vector has multiple sources.

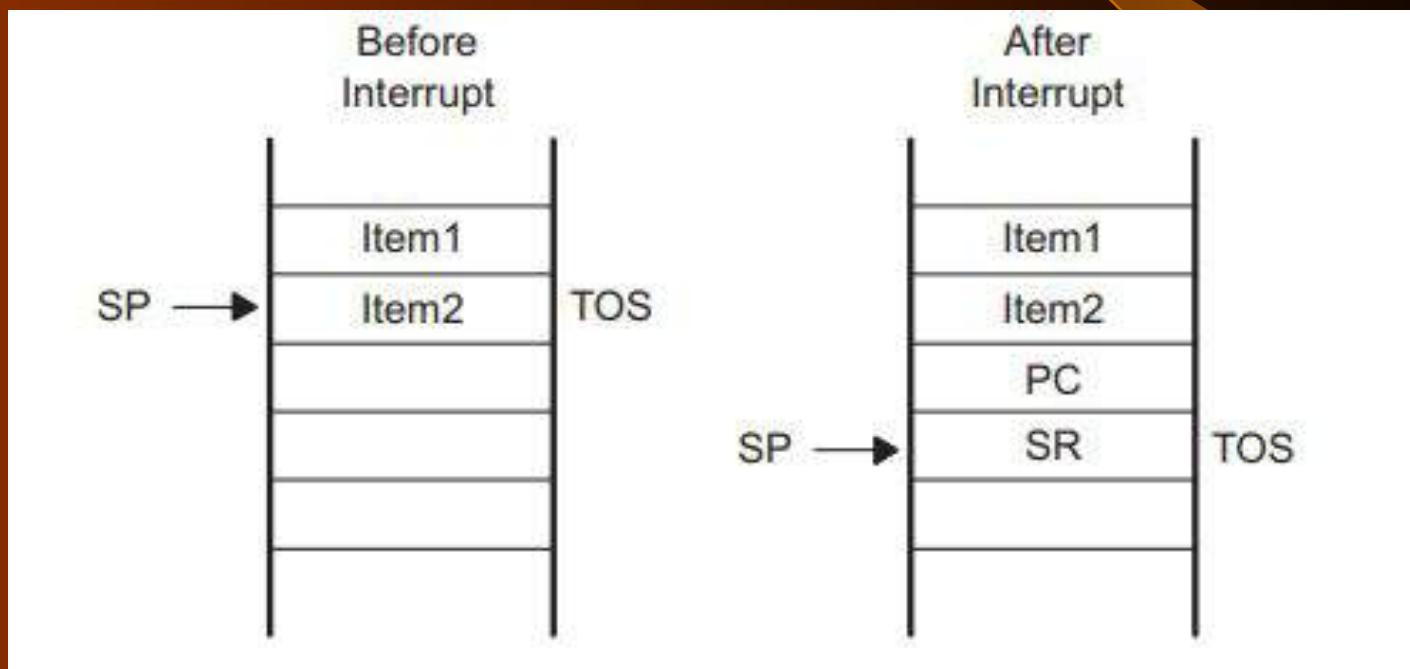
Useful points to remember:

- Keep interrupts short. Avoid delay functions.
- While servicing ISR, other urgent ISRs are disabled unless nested interrupts are allowed.
- If lengthy processing is needed, do minimum in ISR and send a message to main function, by setting a flag.

Interrupt Acceptance

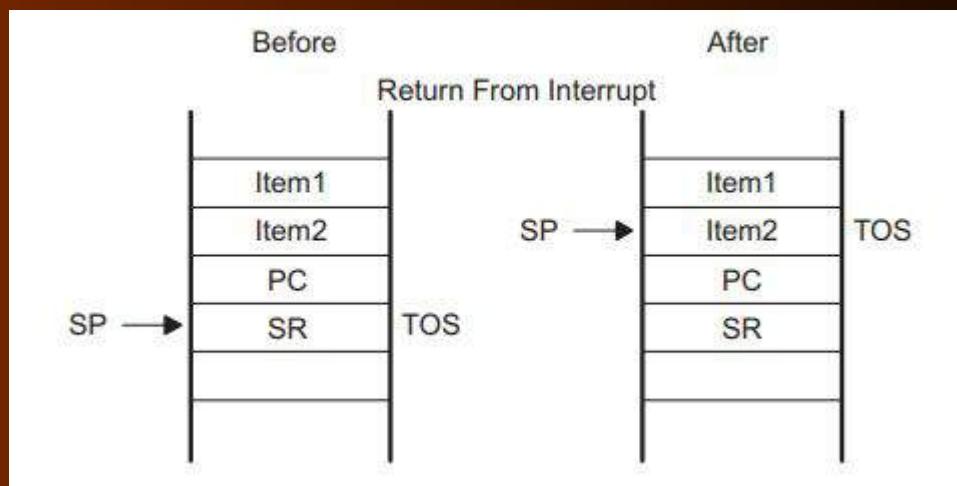
1. Any currently executing instruction is completed.
2. The PC, which points to the next instruction, is pushed onto the stack.
3. The SR(Status Register) is pushed onto the stack.
4. The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
5. The interrupt flag resets automatically on single-source flags.
Multiple source flags remain set for servicing by software.
6. The SR is cleared(for use within the ISR). This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
7. The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.

Interrupt Acceptance (State of the RAM)



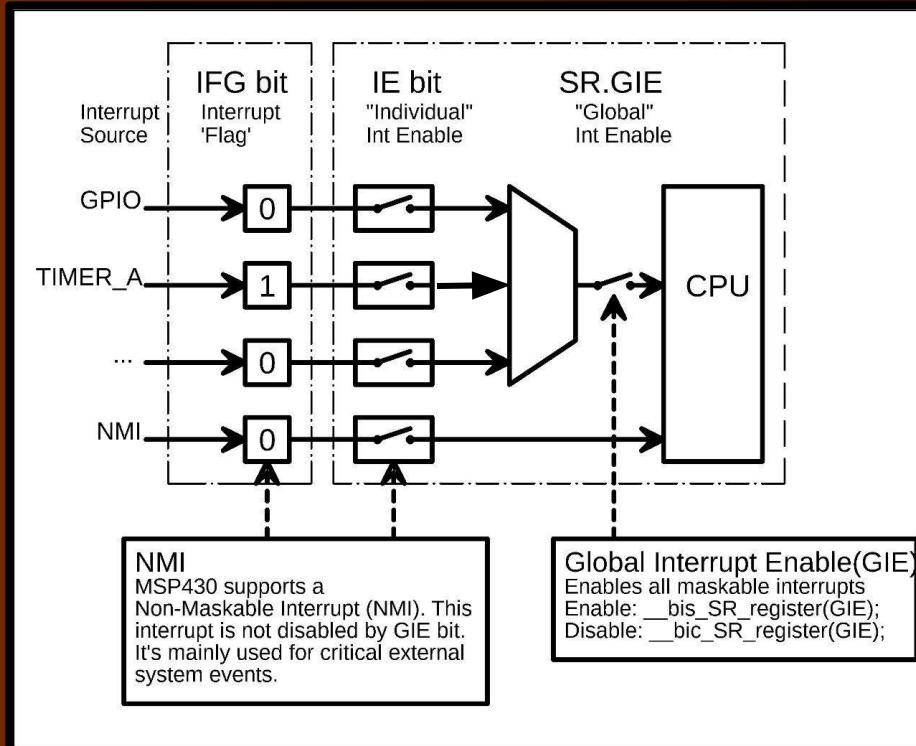
Returning from an Interrupt

1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
2. The PC pops from the stack and begins execution at the point where it was interrupted.



Interrupt Registers for GPIO

- All these registers are 8-bit registers.
- PxIE - Interrupt enable register
- PxIES - Interrupt edge selection register
- PxIFG - Interrupt flag register



PxIE

- PxIE - Interrupt enable register
- 8 Bit Register corresponding to 8 pins on each port.
- Setting any bit as ‘0’ in this register disables the interrupt for the corresponding pin.
- All the bits in this register are 0 by default.
- Setting any bit as ‘1’ in the register enables the interrupt for the corresponding pin.

PxIES

- PxIES - Interrupt edge selection register
- 8 bit register corresponding to 8 pins on each port.
- The bits of this register select the interrupt edge transition type on the corresponding pins.
- Setting a bit as ‘0’ in the register enables interrupt flag when transition is from low to high.
- Setting the bit as ‘1’ in the register enables interrupt flag when transition is from high to low.

PxIFG

- PxIFG - Interrupt flag register
- 8 bit register corresponding to 8 pins on each port.
- A bit of the interrupt flag register is set when the selected interrupt edge occurs on the corresponding pin.
- Using software PxIFG bits can be set or reset.
- PxIFG bits must be reset after the interrupt via software. Setting the PxIFG via software can generate software initiated interrupts.

Interrupt Code Flow

ISR in C

For example:

```
#pragma vector = TIMER0_A0_VECTOR  
__interrupt void CCR0_ISR(void)
```

Some extensions are needed by compiler to differentiate a standard function and an ISR.

- *#pragma* associates the function with a particular interrupt vector. This directive is a special-purpose directive that you can use to turn on or off certain features.
- *__interrupt* keyword in the beginning of the next line that names the function.
- No significance to the name of the function; it is the name of the vector that matters. Name of function is not used in program.
- *__enable_interrupt* in main program sets the GIE bit and turns on interrupts.

ISR names present in CCS for MSP430

(In MSP430.h header file)

```
929 /*****
930 * Interrupt Vectors (offset from 0xFFE0)
931 *****/
932
933 #define VECTOR_NAME(name)           name##_ptr
934 #define EMIT_PRAGMA(x)             _Pragma(#x)
935 #define CREATE_VECTOR(name)        void (* const VECTOR_NAME(name))(void) = &name
936 #define PLACE_VECTOR(vector,section) EMIT_PRAGMA(DATA_SECTION(vector,section))
937 #define ISR_VECTOR(func,offset)    CREATE_VECTOR(func); \
938                           PLACE_VECTOR(VECTOR_NAME(func), offset)
939
940 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
941 #define TRAPINT_VECTOR          ".int00"                      /* 0xFFE0 TRAPINT */
942 #else
943 #define TRAPINT_VECTOR          (0 * 1u)                     /* 0xFFE0 TRAPINT */
944 #endif
945 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
946 #define PORT1_VECTOR            ".int02"                     /* 0xFFE4 Port 1 */
947 #else
948 #define PORT1_VECTOR            (2 * 1u)                     /* 0xFFE4 Port 1 */
949 #endif
950 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
951 #define PORT2_VECTOR            ".int03"                     /* 0xFFE6 Port 2 */
952 #else
953 #define PORT2_VECTOR            (3 * 1u)                     /* 0xFFE6 Port 2 */
954 #endif
955 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
956 #define ADC10_VECTOR             ".int05"                    /* 0xFFEA ADC10 */
957 #else
958 #define ADC10_VECTOR             (5 * 1u)                     /* 0xFFEA ADC10 */
959#endif
```

```
960 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
961 #define USCIAB0TX_VECTOR ".int06" /* 0xFFEC USCI A0/B0 Transmit */
962 #else
963 #define USCIAB0TX_VECTOR (6 * 1u) /* 0xFFEC USCI A0/B0 Transmit */
964 #endif
965 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
966 #define USCIAB0RX_VECTOR ".int07" /* 0xFFEE USCI A0/B0 Receive */
967 #else
968 #define USCIAB0RX_VECTOR (7 * 1u) /* 0xFFEE USCI A0/B0 Receive */
969 #endif
970 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
971 #define TIMER0_A1_VECTOR ".int08" /* 0xFFFF Timer0)A CC1, TA0 */
972 #else
973 #define TIMER0_A1_VECTOR (8 * 1u) /* 0xFFFF Timer0)A CC1, TA0 */
974 #endif
975 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
976 #define TIMER0_A0_VECTOR ".int09" /* 0xFFFF Timer0_A CC0 */
977 #else
978 #define TIMER0_A0_VECTOR (9 * 1u) /* 0xFFFF Timer0_A CC0 */
979 #endif
980 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
981 #define WDT_VECTOR ".int10" /* 0xFFFF Watchdog Timer */
982 #else
983 #define WDT_VECTOR (10 * 1u) /* 0xFFFF Watchdog Timer */
984 #endif
985 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */
986 #define COMPARATORA_VECTOR ".int11" /* 0xFFFF Comparator A */
987 #else
988 #define COMPARATORA_VECTOR (11 * 1u) /* 0xFFFF Comparator A */
989 #endif
```

```
990 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */  
991 #define TIMER1_A1_VECTOR ".int12" /* 0xFFFF8 Timer1_A CC1-4, TA1 */  
992 #else  
993 #define TIMER1_A1_VECTOR (12 * 1u) /* 0xFFFF8 Timer1_A CC1-4, TA1 */  
994 #endif  
995 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */  
996 #define TIMER1_A0_VECTOR ".int13" /* 0xFFFFA Timer1_A CC0 */  
997 #else  
998 #define TIMER1_A0_VECTOR (13 * 1u) /* 0xFFFFA Timer1_A CC0 */  
999 #endif  
1000 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */  
1001 #define NMI_VECTOR ".int14" /* 0xFFFC Non-maskable */  
1002 #else  
1003 #define NMI_VECTOR (14 * 1u) /* 0xFFFC Non-maskable */  
1004 #endif  
1005 #ifdef __ASM_HEADER__ /* Begin #defines for assembler */  
1006 #define RESET_VECTOR ".reset" /* 0xFFFFE Reset [Highest Priority] */  
1007 #else  
1008 #define RESET_VECTOR (15 * 1u) /* 0xFFFFE Reset [Highest Priority] */  
1009 #endif  
1010
```

What should be done in an ISR

- Save system context (done automatically)
- Re-enable interrupts if required.
- The interrupt code
- If multi-source interrupt, then read associated register to determine source and clear the flag.
- Restore system context (done automatically)
- Return

Variables for Interrupts

- Volatile: By declaring a variable as volatile, the compiler gets to know that the value of the variable is susceptible to frequent changes (with no direct action of the program) and hence the compiler does not keep a copy of the variable in a register (like cache). This prevents the program to misinterpret the value of this variable.
- As a thumb rule: variables associated with input ports and interrupts should be declared as volatile.

Check List

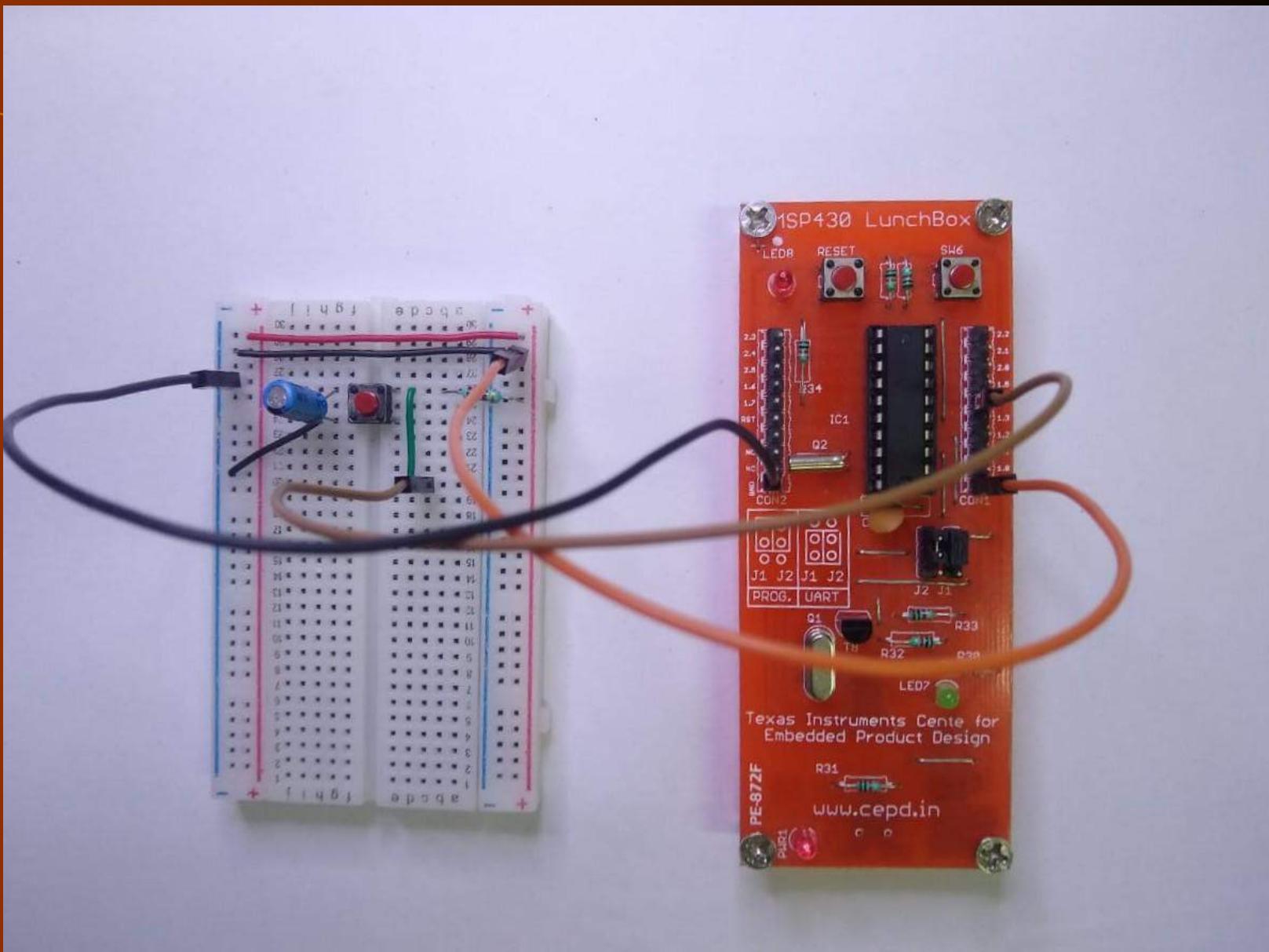
- Enabled interrupts both in their module and generally (GIE bit)?
- Provided ISR for all the enabled interrupts?
- Included code to acknowledge interrupt that share a vector even if only 1 source is active?
- Are the variables declared as volatile?

Switch Interfacing using Interrupts

Example Code 1(Bad ISR): HelloInterrupt

```
1 #include <msp430.h>
2
3 #define SW      BIT3           // Switch -> P1.3
4 #define RED     BIT7           // Green LED -> P1.7
5
6 /*@brief entry point for the code*/
7 void main(void) {
8     WDTCTL = WDTPW | WDTHOLD;          // Stop watchdog timer
9
10    P1DIR |= RED;                   // Set LED pin -> Output
11    P1DIR &= ~SW;                  // Set SW pin -> Input
12    P1REN |= SW;                  // Enable Resistor for SW pin
13    P1OUT |= SW;                  // Select Pull Up for SW pin
14
15    P1IES &= ~SW;                  // Select Interrupt on Rising Edge
16    P1IE |= SW;                  // Enable Interrupt on SW pin
17
18    __bis_SR_register(GIE);        // Enable CPU Interrupt
19
20    while(1);
21}
22
23 /*@brief entry point for switch interrupt*/
24 #pragma vector=PORT1_VECTOR
25 _interrupt void Port_1(void)
26{
27    if(P1IFG & SW)                // If SW is Pressed
28    {
29        P1OUT ^= RED;            // Toggle RED LED
30        volatile unsigned long i;
31        for(i = 0; i<10000; i++); //delay
32        P1IFG &= ~SW;           // Clear SW interrupt flag
33    }
34
35}
36
```

External Switch



Example Code 2(Better ISR): HelloInterrupt_Rising

```
1 #include <msp430.h>
2
3 #define SW      BIT4          // Switch -> P1.4
4 #define RED     BIT7          // Green LED -> P1.7
5
6 /*@brief entry point for the code*/
7 void main(void) {
8     WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer
9
10    P1DIR |= RED;                     // Set LED pin -> Output
11    P1DIR &= ~SW;                    // Set SW pin -> Input
12    P1REN |= SW;                     // Enable Resistor for SW pin
13    P1OUT |= SW;                     // Select Pull Up for SW pin
14
15    P1IES &= ~SW;                   // Select Interrupt on Rising Edge
16    P1IE |= SW;                     // Enable Interrupt on SW pin
17
18    __bis_SR_register(GIE);          // Enable CPU Interrupt
19
20    while(1);
21}
22
23 /*@brief entry point for switch interrupt*/
24 #pragma vector=PORT1_VECTOR
25 __interrupt void Port_1(void)
26{
27    if(P1IFG & SW)                  // If SW is Pressed
28    {
29        P1OUT ^= RED;              // Toggle RED LED
30        P1IFG &= ~SW;             // Clear SW interrupt flag
31    }
32}
```

Example Code 3: HelloInterrupt_Falling

```
1 #include <msp430.h>
2
3 #define SW      BIT4          // Switch -> P1.4
4 #define RED     BIT7          // Green LED -> P1.7
5
6 /*@brief entry point for the code*/
7 void main(void) {
8     WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer
9
10    P1DIR |= RED;                     // Set LED pin -> Output
11    P1DIR &= ~SW;                    // Set SW pin -> Input
12    P1REN |= SW;                     // Enable Resistor for SW pin
13    P1OUT |= SW;                     // Select Pull Up for SW pin
14
15    P1IES |= SW;                     // Select Interrupt on Falling Edge
16    P1IE |= SW;                     // Enable Interrupt on SW pin
17
18    __bis_SR_register(GIE);          // Enable CPU Interrupt
19
20    while(1);
21}
22
23 /*@brief entry point for switch interrupt*/
24 #pragma vector=PORT1_VECTOR
25 __interrupt void Port_1(void)
26{
27    if(P1IFG & SW)                  // If SW is Pressed
28    {
29        P1OUT ^= RED;              // Toggle RED LED
30        P1IFG &= ~SW;             // Clear SW interrupt flag
31    }
32}
33
```

Exercise

Modify the existing code to toggle LED1 when SW1 is pressed and to toggle LED2 when SW2 is pressed.



Thank you!

Introduction to Embedded System Design

Seven Segment Displays with MSP430, Low Power Modes in MSP430

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

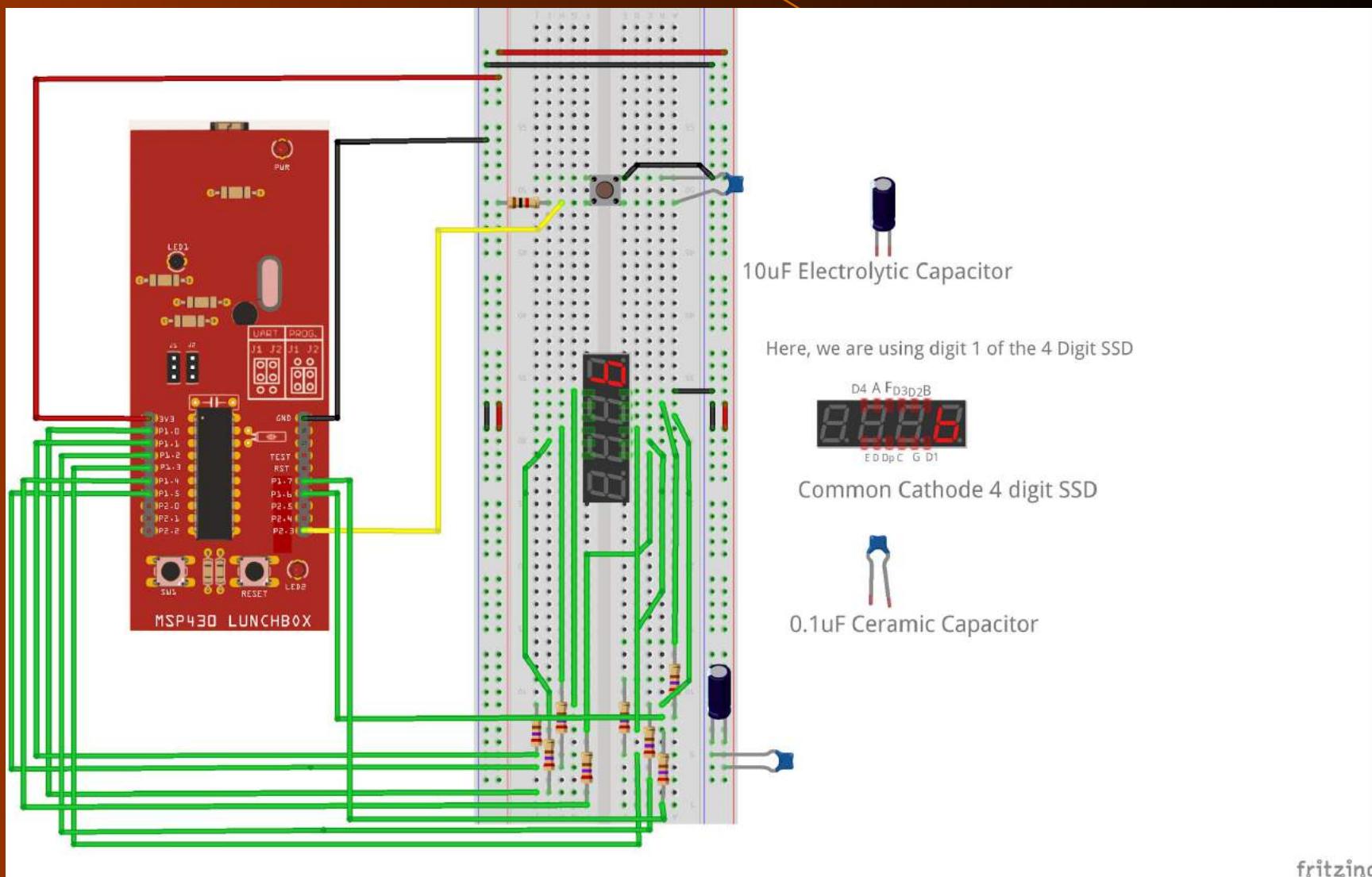
Interfacing SSD with MSP430

HelloSSD:

This example code is used to display a hexadecimal (0 to F) up counter on a single digit SSD(Common Cathode). The count increases on every press of an external switch.

Interfacing SSD with MSP430

Fritzing Diagram



Code Example: HelloSSD

```
1 #include <msp430.h>
2
3 #define SW      BIT3           // Switch -> P2.3
4
5 // Define Pin Mapping of 7-segment Display
6 // Segments are connected to P1.0 - P1.7
7 #define SEG_A    BIT0
8 #define SEG_B    BIT1
9 #define SEG_C    BIT2
10 #define SEG_D   BIT3
11 #define SEG_E   BIT4
12 #define SEG_F   BIT5
13 #define SEG_G   BIT6
14 #define SEG_DP  BIT7
15
16 // Define each digit according to truth table
17 #define D0  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F)
18 #define D1  (SEG_B + SEG_C)
19 #define D2  (SEG_A + SEG_B + SEG_D + SEG_E + SEG_G)
20 #define D3  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_G)
21 #define D4  (SEG_B + SEG_C + SEG_F + SEG_G)
22 #define D5  (SEG_A + SEG_C + SEG_D + SEG_F + SEG_G)
23 #define D6  (SEG_A + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
24 #define D7  (SEG_A + SEG_B + SEG_C)
25 #define D8  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
26 #define D9  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_F + SEG_G)
27 #define DA  (SEG_A + SEG_B + SEG_C + SEG_E + SEG_F + SEG_G)
28 #define DB  (SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
29 #define DC  (SEG_A + SEG_D + SEG_E + SEG_F)
30 #define DD  (SEG_B + SEG_C + SEG_D + SEG_E + SEG_G)
31 #define DE  (SEG_A + SEG_D + SEG_E + SEG_F + SEG_G)
32 #define DF  (SEG_A + SEG_E + SEG_F + SEG_G)
33
34
```

```
34
35 // Define mask value for all digit segments except DP
36 #define DMASK ~(SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
37
38 // Store digits in array for display
39 const unsigned int digits[16] = {D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, DA, DB, DC, DD, DE, DF};
40
41 volatile unsigned int i = 0;
42
43 /*@brief entry point for the code*/
44 void main(void) {
45     WDTCTL = WDTPW | WDTHOLD;           //! Stop Watch dog
46
47     // Initialize 7-segment pins as Output
48     P1DIR |= (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E+ SEG_F + SEG_G + SEG_DP);
49
50     P2DIR &= ~SW;                      // Set SW pin -> Input
51
52     while(1)
53     {
54         if(!(P2IN & SW))             // If SW is Pressed
55         {
56             __delay_cycles(20000);    //Delay to avoid Switch Bounce
57             while(!(P2IN & SW));   // Wait till SW Released
58             __delay_cycles(20000);    //Delay to avoid Switch Bounce
59             i++;                   //Increment count
60             if(i>15)
61             {
62                 i=0;
63             }
64         }
65         P1OUT = (P1OUT & DMASK) + digits[i]; // Display current digit
66     }
67 }
```

Low Power Modes in MSP430

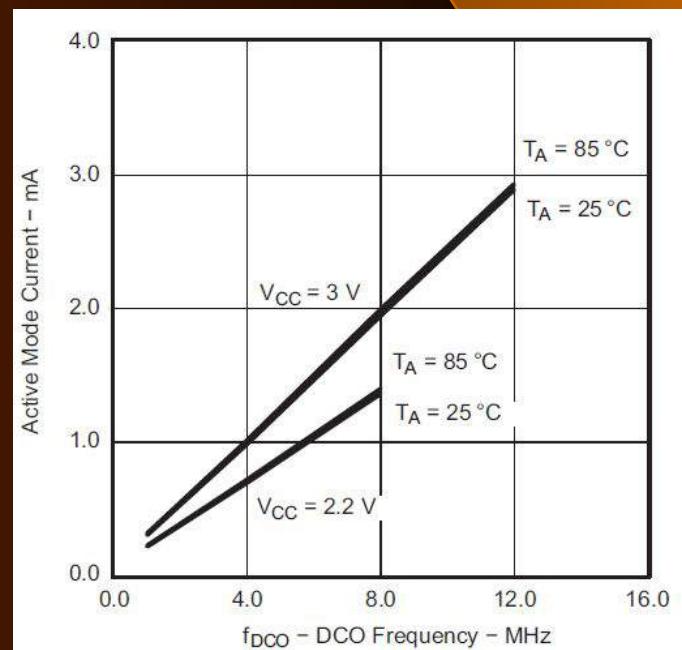
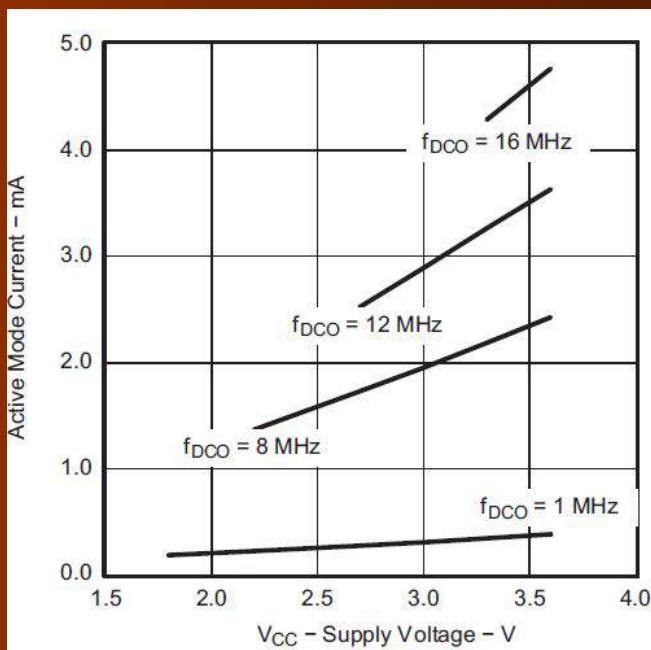
The MSP430 was designed primarily for low power applications and this is reflected in a range of low-power modes of operation.

OPERATING MODES:

- Active Mode
- LPM0
- LPM1
- LPM2
- LPM3
- LPM4

Active Mode

- All clocks are active.
- SUPPLY CURRENT(at 1MHz):
 - 230uA(at 2.2V)
 - 330uA(at 3.0V)



LPM0

- CPU is disabled.
- ACLK and SMCLK remain active, MCLK is disabled.
- Supply current(at 1 MHz) : 56uA(at 2.2V)

LPM1

- CPU is disabled.
- ACLK and SMCLK remain active, MCLK is disabled.
- DCO and DC generator is disabled if DCO is not used for SMCLK

LPM2

- CPU is disabled.
- MCLK and SMCLK are disabled.
- DCO's DC generator remains enabled.
- ACLK remains active.
- Supply Current : 22uA(at 2.2V)

LPM3

- CPU is disabled.
- MCLK and SMCLK are disabled.
- DCO's DC generator is disabled.
- ACLK remains active.
- Supply Current : 0.5uA(at 2.2V)

LPM4

- CPU is disabled.
- ACLK is disabled.
- MCLK and SMCLK are disabled.
- DCO's DC generator is disabled.
- Crystal oscillator is stopped.
- Supply Current : 0.1uA(at 2.2V)

Controlling Low Power Modes

- LPM operations are controlled through 4 bits of the Status Register: SCG0, SCG1, CPUOFF and OSCOFF.
- All these bits are cleared in active mode and particular combinations are set for each LPM.

Controlling Low Power Modes

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks disabled

- Two ways to put MSP430 into LPM Mode 4, for example, are:
 - `_low_power_mode_4();` //This also enables the GIE bit.
 - `_bis_SR_register(LPM4_bits + GIE);` //+GIE is used to enable interrupts.

Interrupts and LPM

- MSP430 is designed to stay in a Low Power Mode for most of the time.
- The main function **generally** configures the peripherals, enables interrupts, puts the MSP430 into LPM, and plays no further role.
- MSP430 is woken up only when an interrupt comes.
- An LPM is suspended whenever an enabled interrupt is serviced.
- On entering the ISR, the SR is stored on stack and the CPUOFF, SCG1, OSCOFF bits are automatically reset.
- On exiting the ISR, the original SR is popped from the stack and the previous operating mode is restored, so that the LPM is resumed after the ISR.

Ultra Low Power Mode Advisor

- ULP Mode Advisor is integrated into CCS.
- It checks your code against a thorough checklist to achieve the lowest power possible and provides detailed notifications and remarks

Code Example: HelloLPM

```
1 #include <msp430.h>
2
3 #define SW      BIT3           // Switch -> P1.3
4 #define RED     BIT7           // Red LED -> P1.7
5
6 /*@brief entry point for the code*/
7 void main(void) {
8     WDTCTL = WDTPW | WDTHOLD;          // Stop Watch dog timer
9
10    P1DIR |= RED;                  // Set LED pin -> Output
11    P1OUT &= ~RED;                // Turn RED LED off
12
13    P1DIR &= ~SW;                 // Set SW pin -> Input
14    P1IES &= ~SW;                // Select Interrupt on Rising Edge
15    P1IE |= SW;                 // Enable Interrupt on SW pin
16
17    unsigned int i;
18
19    while(1)
20    {
21        __bis_SR_register(LPM4_bits + GIE); // Enter LPM4 and Enable CPU Interrupt
22
23        P1OUT ^= RED;                // Toggle RED LED
24        for(i=0; i<20000; i++);
25    }
26}
27
28 /*@brief entry point for switch interrupt*/
29 #pragma vector=PORT1_VECTOR
30 __interrupt void Port_1(void)
31{
32    __bic_SR_register_on_exit(LPM4_bits + GIE); // Exit LPM4 on return to main
33    P1IFG &= ~SW;                   // Clear SW interrupt flag
34}
```



Thank you!

Introduction to Embedded System Design

Interfacing Liquid Crystal Displays with MSP430

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Liquid Crystal Displays

- Numeric
- Alphanumeric
- Character
- Graphics

Liquid Crystal Displays

- Liquid crystal display technology works by manipulating light.
- Uses two polarizers - horizontal and vertical placed on top of each other, with a source of light at the bottom.
- Liquid crystal between the two polarizers
- Liquid Crystals, when potential is applied, have the ability to ‘twist’ the light.

Liquid Crystal Display Types

- Character LCD Type: 8*1, 8*2, 12*2, 16*1, 16*2, 16*4, 20*1, 20*2, 20*4, 40*1



8*2



12*2



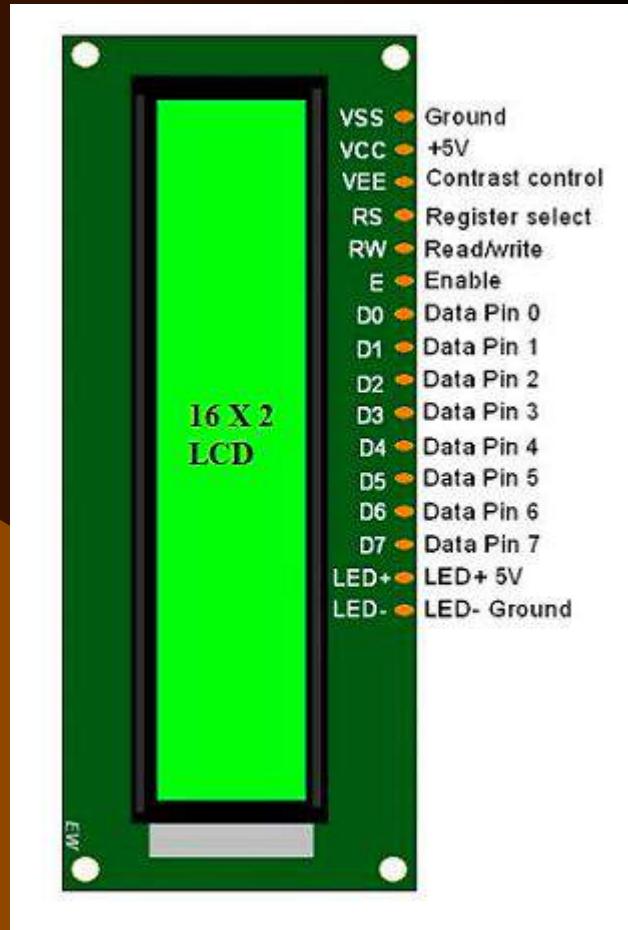
16*2



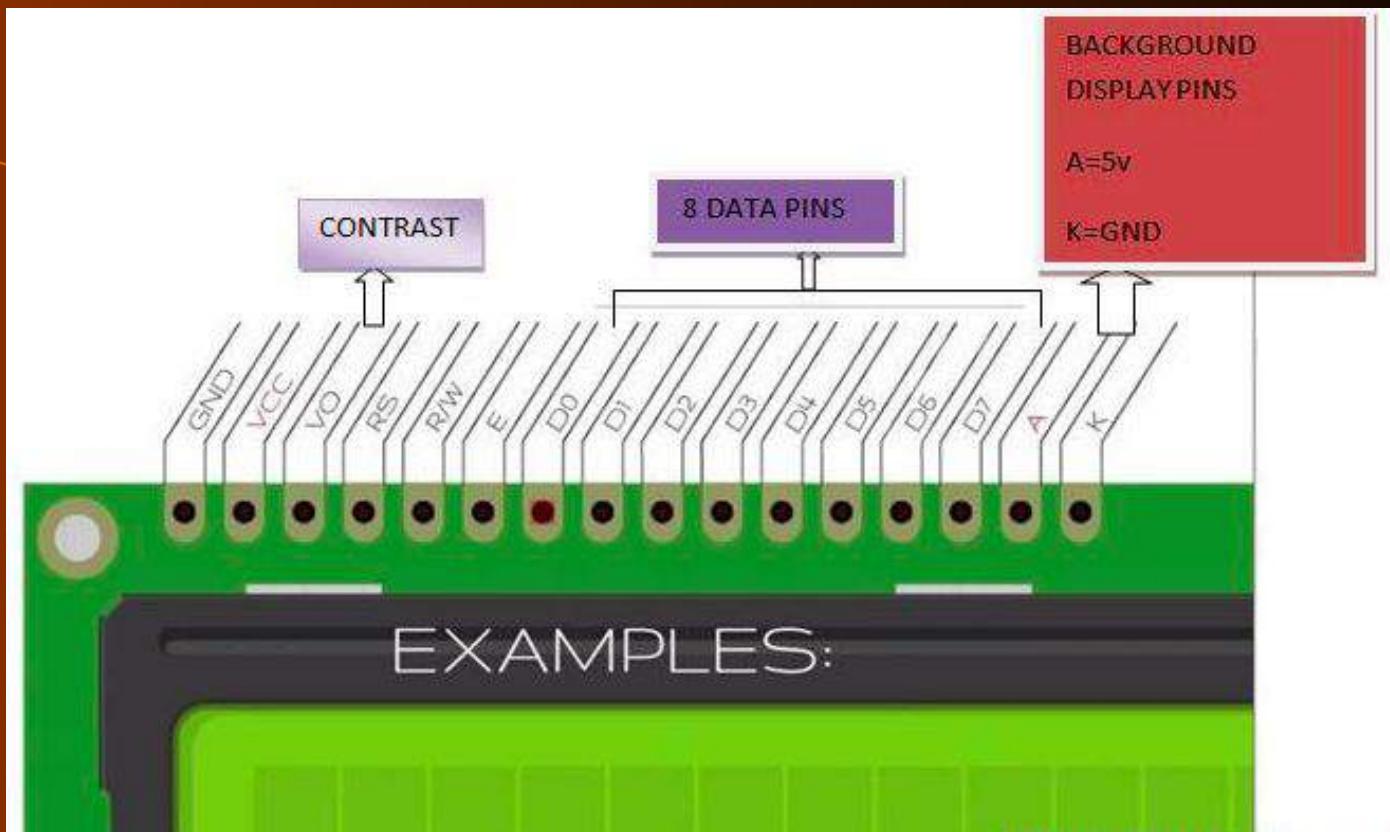
20*1

Hitachi HD44780 LCD Controller

- In this course, we will use the 16×2 LCD display, which is a very basic module (16-pin) commonly used in DIYs circuits.
- The 16×2 translates to a display 16 characters per line in 2 such lines.
- In LCD displays, each character is in a 5×8 matrix. Where 5 are the number of columns and 8 is the number of rows. This knowledge will be used when we have to print custom text or letters.



Pinout for 16X2 LCD



As shown in the figure above, it has a pin for contrast, 8 Data pins along with pins for GND and VCC. R/W is Read Write pin. RS is the Register Select pin. E is the Enable pin.

16X2 LCD Pin Description:

PIN		Description
<ul style="list-style-type: none"> • Ground Pin (GND) and • Supply Pin (4.7V - 5.3V) 	GND VCC	Grounded pin and Vcc is given on the respective pin.
<ul style="list-style-type: none"> • Read/Write and • An Enable pin (-ve Edge triggered for Write and +ve edge triggered for Read) 	R/W* E	<ol style="list-style-type: none"> 1. Low to write to the register. High to read from the register 2. An edge triggered signal used to writing or reading data to/from LCD
<ul style="list-style-type: none"> • Contrast adjustment. A variable resistor (mostly a preset) is generally attached on this pin. 	VO/VEE	Output of the potentiometer is connected to this pin. Rotate the potentiometer knob forward and backwards to adjust the LCD contrast.
<ul style="list-style-type: none"> • 8 Data pins 	D0-D7	8 data pins for data transfer.
<ul style="list-style-type: none"> • Register Select: (A 16X2 LCD has two registers, namely, command and data.) 	RS	The register select is used to switch from one register to other. RS=0 for command register, whereas RS=1 for data register.
<ul style="list-style-type: none"> • Backlit Supply (5V) 	LED+	
<ul style="list-style-type: none"> • Backlit Ground (GND) 	LED-	

Command Codes for LCD

S.no	Hex Code	Command to LCD Instruction Register			
1.	01	Clear Display Screen	11.	0F	Display On, Cursor Blinking
2.	02	Return Home	12.	10	Shift Cursor to Left
3.	04	Decrement Cursor(shift to left)	13.	14	Shift Cursor to Right
4.	06	Increment Cursor	14.	18	Shift Entire display to the Left
5.	05	Shift Display Right	15.	1C	Shift Entire display to the Right
6.	07	Shift Display Left	16.	80	Force Cursor to Beginning (1st line)
7.	08	Display Off, Cursor Off	17.	C0	Force Cursor to Beginning (2nd line)
8.	0A	Display Off, Cursor On	18.	38	2 line and 5X7 Matrix
9.	0C	Display On, Cursor Off	19.	28	2 line 5X7 matrix in 4bit mode
10.	0E	Display Off, Cursor On	20.	32	Send for 4bit initialisation of LCD
			21.	33	Send for 4bit initialisation of LCD

16*2 LCD Working Modes

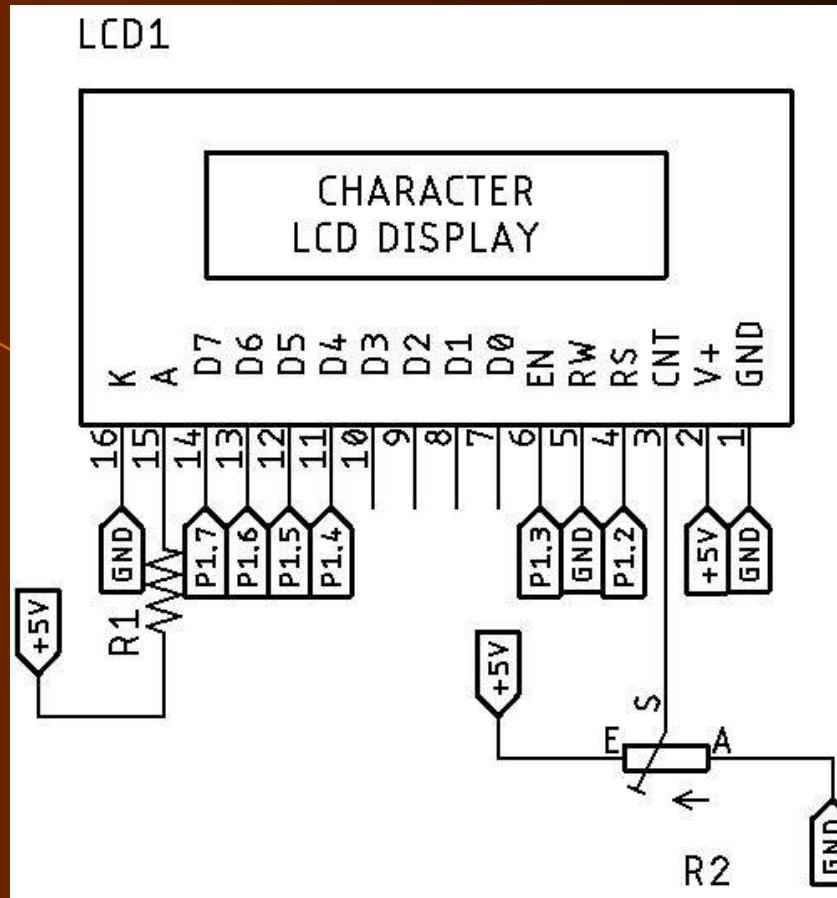
- 16*2 character display can be operated in two modes -
 - 8 Bit mode - This is the default mode in which all the data pins (D0 - D7) are used.
 - 4 Bit mode - Only four data pins (D4 - D7) are used.

4 Bit mode

- In 4-bit mode, data/command is sent in 4-bit (nibble) format.
- To do this 1st send Higher 4-bit and then send lower 4-bit of data / command.
- Only 4 data (D4 - D7) pins of 16x2 of LCD are connected to microcontroller and other control pins RS (Register select), RW (Read / write), E (Enable) are connected to other GPIO Pins of controller.

```
LCD_Command(0x33);  
LCD_Command(0x32); /* Send for 4 bit initialization of LCD */  
LCD_Command(0x28); /* 2 line, 5*7 matrix in 4-bit mode */  
LCD_Command(0x0C); /* Display on cursor off */  
LCD_Command(0x06); /* Increment cursor (shift cursor to right) */  
LCD_Command(0x01); /* Clear display screen */
```

LCD Connections



- Data pins, EN, RW connected to LunchBox
- +5V from external power supply.
Remember to connect GND of +5V supply and the GND of LunchBox.

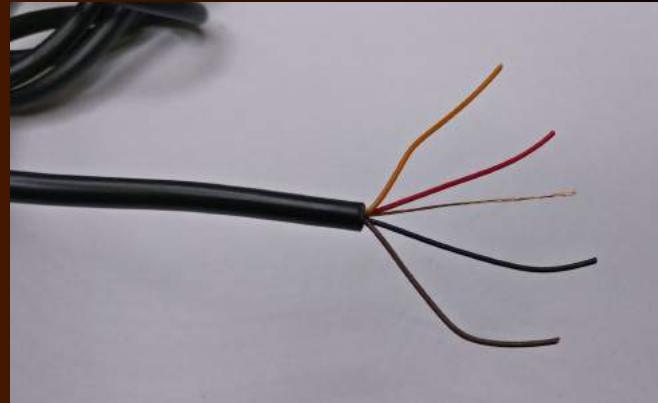
+5V Power Supply Options

1. Modified USB cable
2. Breadboard Power Supply
3. Lab Bench Power Supply

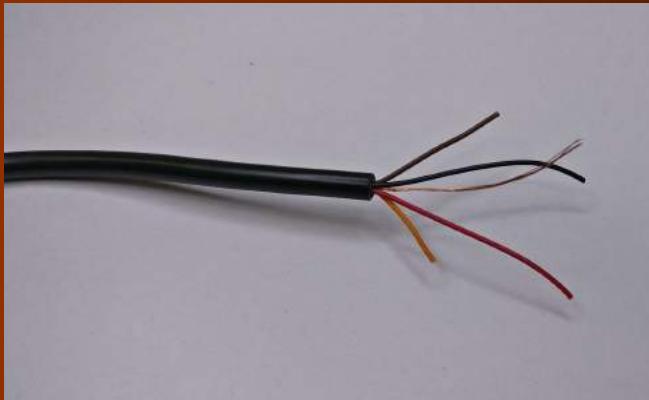
Modifying the USB Cable



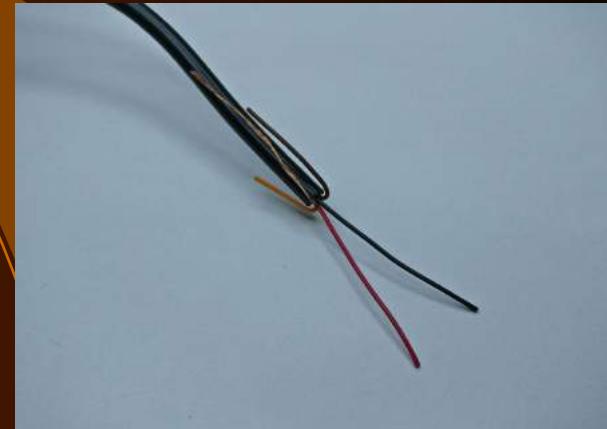
Step1: Cut the end (other than the USB Type A male end) of the cable.



Step2: Strip the insulator part of the cable to see five different wires.

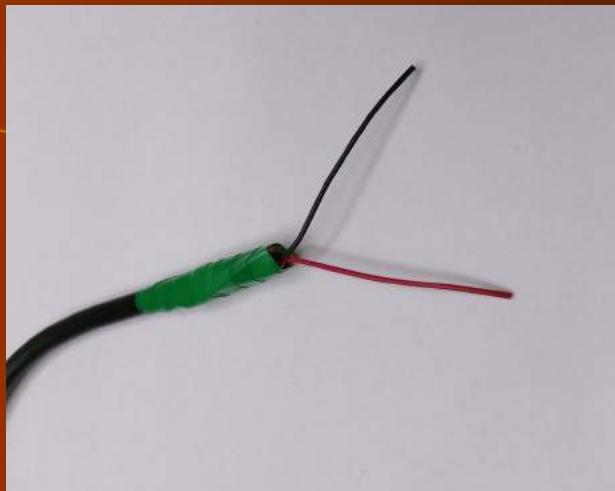


Step3: The Red and Black ones are used for the Power supply. Cut the three other wires at different lengths.

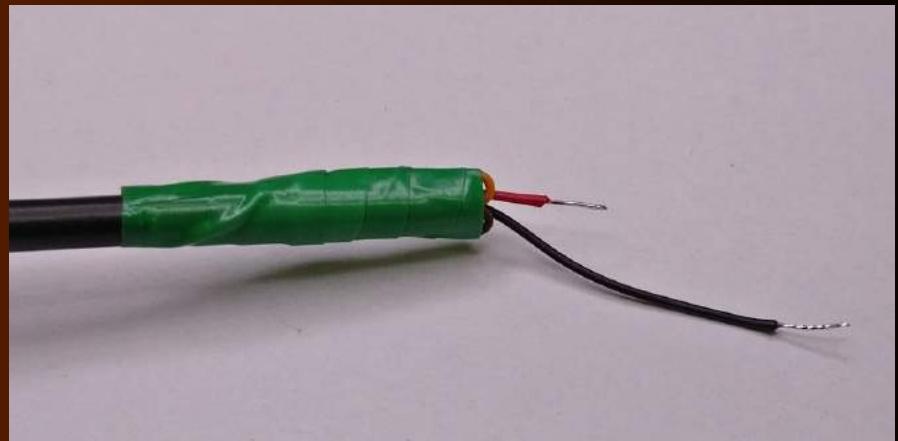


Step4: Now, bend the three wires.

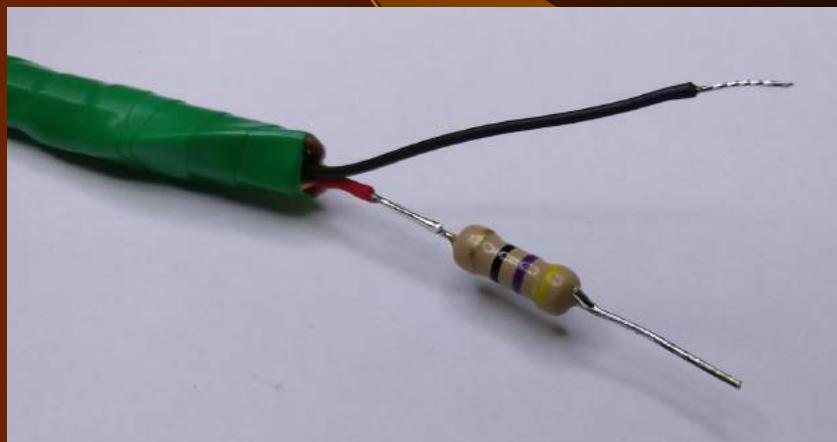
Modifying the USB Cable



Step5: Tape the three ends with the help of an insulation tape.



Step6: Cut the red wire short. Now, strip the ends of the red and black wires and use a soldering iron to tin the ends.



Step7: Solder one end of a 47 ohm resistor with the Red wire.

The ‘Hello LCD’ Code

```
1 #include <msp430.h>
2 #include <inttypes.h>
3
4 #define CMD      0
5 #define DATA     1
6
7 #define LCD_OUT   P1OUT
8 #define LCD_DIR   P1DIR
9 #define D4        BIT4
10 #define D5       BIT5
11 #define D6       BIT6
12 #define D7       BIT7
13 #define RS        BIT2
14 #define EN       BIT3
15
16 /**
17 */
18 * @brief Delay function for producing delay in 0.1 ms increments
19 * @param t milliseconds to be delayed
20 * @return void
21 */
22 void delay(uint8_t t)
23 {
24     uint8_t i;
25     for(i=t; i > 0; i--)
26         __delay_cycles(100);
27 }
```

We have used uint8_t, over here, because our variable is such that its value will not exceed 255 in delay function

In this code, uint8_t datatype is used to declare the variable. uint8 is used to write implementation-independent code.

unsigned char is not guaranteed to be an 8-bit type in many implementation but. uint8_t is. uint8 is defined in inttypes.h header file.

```

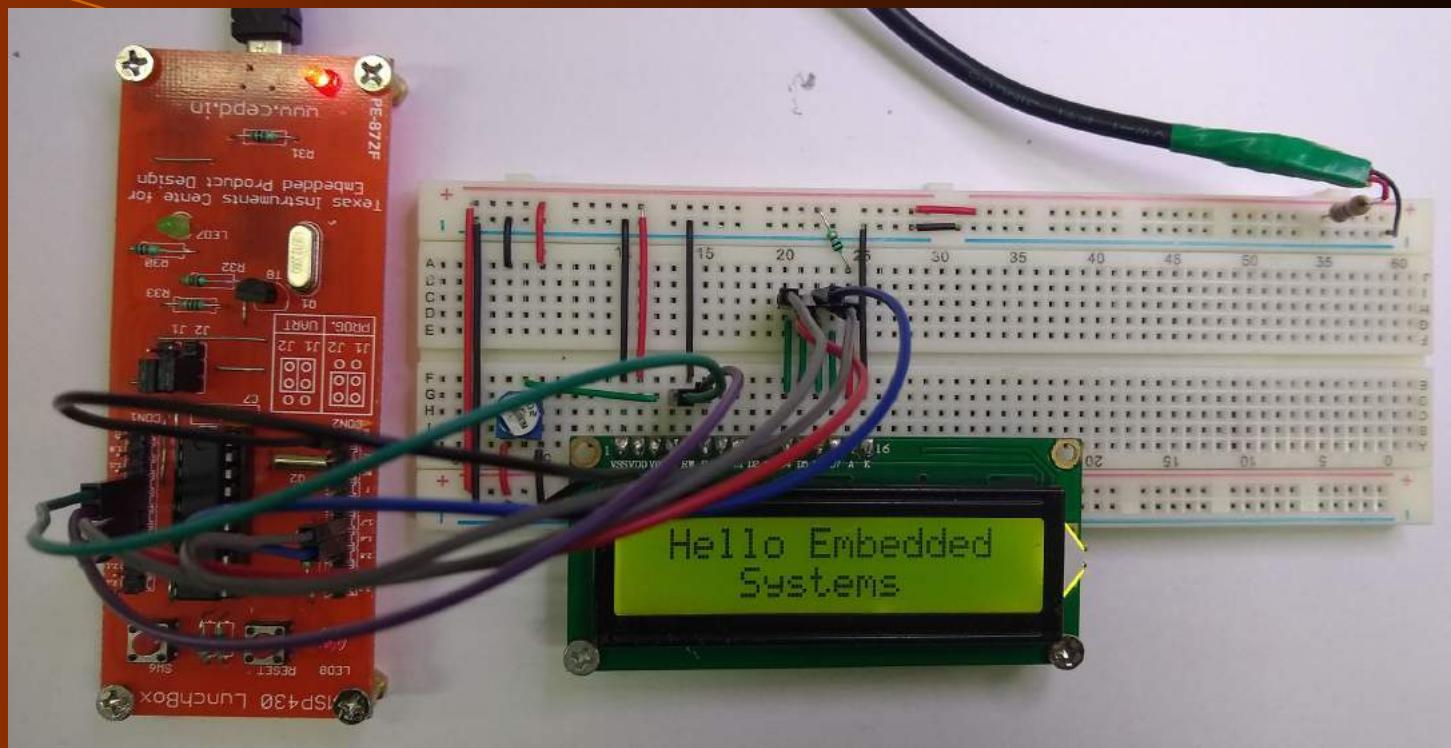
28
29 /**
30 * @brief Function to pulse EN pin after data is written
31 * @return void
32 */
33 void pulseEN(void)
34 {
35     LCD_OUT |= EN;
36     delay(1);
37     LCD_OUT &= ~EN;
38     delay(1);
39 }
40
41 /**
42 * @brief Function to write data/command to LCD
43 * @param value Value to be written to LED
44 * @param mode Mode -> Command or Data
45 * @return void
46 */
47 void lcd_write(uint8_t value, uint8_t mode)
48 {
49     if(mode == CMD)
50         LCD_OUT &= ~RS;           // Set RS -> LOW for Command mode
51     else
52         LCD_OUT |= RS;          // Set RS -> HIGH for Data mode
53
54     LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));           // Write high nibble first
55     pulseEN();
56     delay(1);
57
58     LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));    // Write low nibble next
59     pulseEN();
60     delay(1);
61 }
```

We know any character will be less than (0-255)
 2^8 value in its ASCII notation.

```
62
63  /**
64   *@brief Function to print a string on LCD
65   *@param *s pointer to the character to be written.
66   *@return void
67   */
68  void lcd_print(char *s)
69  {
70      while(*s)
71      {
72          lcd_write(*s, DATA);
73          s++;
74      }
75  }
76
77 /**
78  *@brief Function to move cursor to desired position on LCD
79  *@param row Row Cursor of the LCD
80  *@param col Column Cursor of the LCD
81  *@return void
82  */
83  void lcd_setCursor(uint8_t row, uint8_t col)
84  {
85      const uint8_t row_offsets[] = { 0x00, 0x40};
86      lcd_write(0x80 | (col + row_offsets[row]), CMD);
87      delay(1);
88  }
89
```

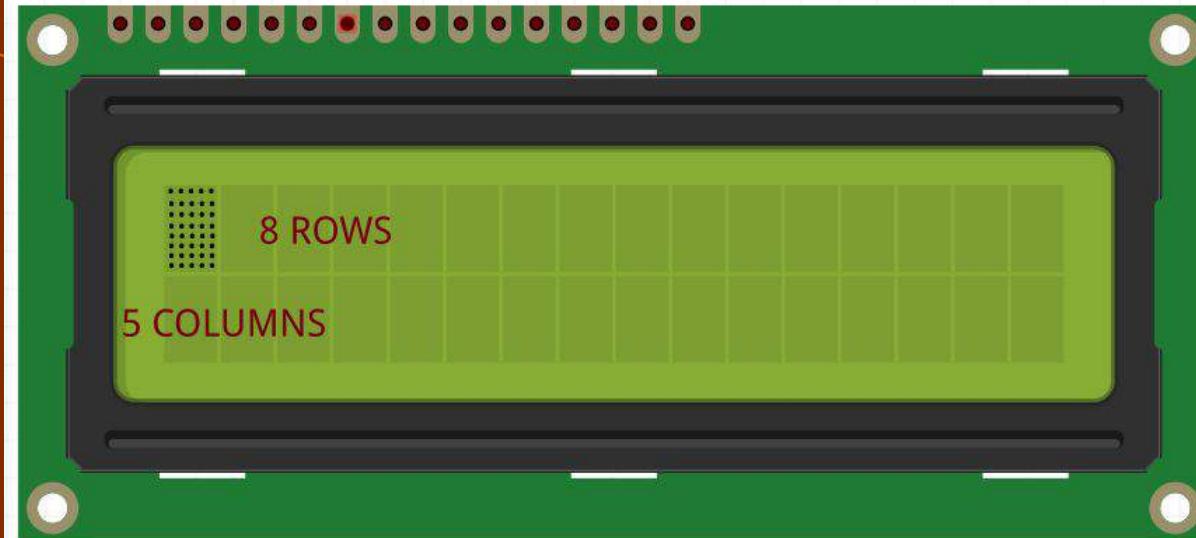
```
90  /**
91   *@brief Initialize LCD
92   */
93 void lcd_init()
94 {
95     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
96     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
97
98     delay(150);                      // Wait for power up ( 15ms )
99     lcd_write(0x33, CMD);            // Initialization Sequence 1
100    delay(50);                      // Wait ( 4.1 ms )
101    lcd_write(0x32, CMD);            // Initialization Sequence 2
102    delay(1);                       // Wait ( 100 us )
103
104    // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
105
106    lcd_write(0x28, CMD);            // 4 bit mode, 2 line
107    delay(1);
108
109    lcd_write(0x0C, CMD);            // Display ON, Cursor OFF, Blink OFF
110    delay(1);
111
112    lcd_write(0x01, CMD);            // Clear screen
113    delay(20);
114
115    lcd_write(0x06, CMD);            // Auto Increment Cursor
116    delay(1);
117
118    lcd_setCursor(0,0);              // Goto Row 1 Column 1
119 }
120
```

```
120
121  /*@brief entry point for the code*/
122  void main(void)
123  {
124      WDTCTL = WDTPW + WDTHOLD;      //! Stop Watchdog (Not recommended for code in production and devices working in field)
125
126      lcd_init();
127      lcd_setCursor(0,1);
128      lcd_print("Hello Embedded");
129      lcd_setCursor(1,5);
130      lcd_print("Systems!");
131      while(1);
132 }
```



Custom Display on the LCD

In LCD displays, each character is in a 5×8 matrix.



Where 5 are the number of columns and 8 is the number of rows.

CG-RAM is the main component in making custom characters. It stores the custom characters once declared in the code.

- CG-RAM size is 64 byte providing the option of creating eight characters at a time. Each character is eight byte in size.
- CG-RAM address starts from 0x40 (Hexadecimal) or 64 in decimal.
 - We can generate custom characters at these addresses.
- Once we generate our characters at these addresses, now we can print them on the LCD at any time by just sending simple commands to the LCD as shown on the right.

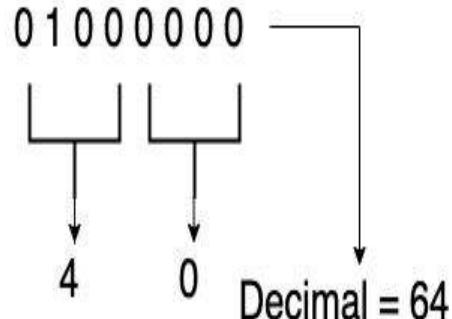
CG-RAM Characters	CG-RAM Address (Hexadecimal)	Commands to display Generated Characters
1 st Character	0x40	0
2 nd Character	0x48	1
3 rd Character	0x56	2
4 th Character	0x64	3
5 th Character	0x72	4
6 th Character	0x80	5
7 th Character	0x88	6
8 th Character	0x96	7

In the table above, you can see starting addresses for each character with their printing commands.

The first character is generated at address 0x40 to 0x47 and is printed on LCD by just sending simple command 0 to the LCD. The second character is generated at address 0x48 to 0x55 and is printed by sending 1 to LCD.

Display letter ‘b’ on the LCD

Starting Address of CG-RAM



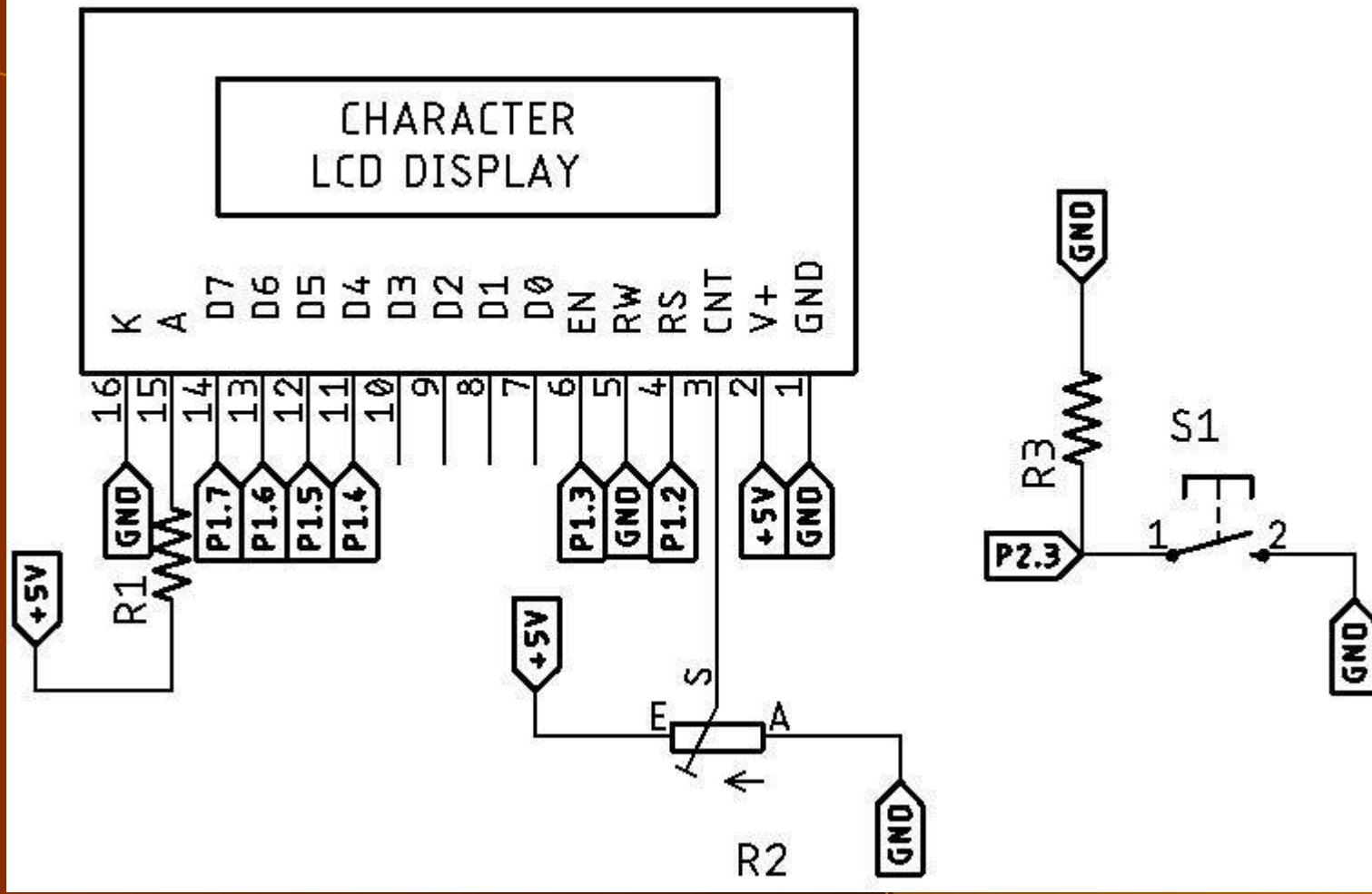
Byte Address	C1	C2	C3	C4	C5	Pattern For 'b' is:
0 0 0 0	1	0	0	0	0	Row1 0x10
0 0 0 1	1	0	0	0	0	Row2 0x10
0 0 1 0	1	0	1	1	0	Row3 0x16
0 0 1 1	1	1	0	0	1	Row4 0x19
0 1 0 0	1	0	0	0	1	Row5 0x11
0 1 0 1	1	0	0	0	1	Row6 0x11
0 1 1 0	1	1	1	1	0	Row7 0x1E
0 1 1 1	0	0	0	0	0	<-Cursor Position

Here, in the 8 bit data,

- First 3 bits are treated as Don't cares.
- Rest 5 bits are loaded at the address

- Send address where you want to create character. Now create your character at this address.
- Send the ‘b’ character array values defined above one by one to the data register of LCD.
- To print the generated character at 0x40.
- Send command 0 to command register of LCD.

LCD1



The ‘Hello LCD With Custom Character’ Code

```
1 #include <msp430.h>
2 #include <inttypes.h>
3
4 #define CMD      0
5 #define DATA     1
6
7 #define LCD_OUT   P1OUT
8 #define LCD_DIR   P1DIR
9 #define D4        BIT4
10 #define D5       BIT5
11 #define D6       BIT6
12 #define D7       BIT7
13 #define RS        BIT2
14 #define EN       BIT3
15
16 #define SW       BIT3
17
18 #define LCD_SETGRAMADDR 0x40
19
20 //Heart character
21 uint8_t heart[8] = {
22   0x00,
23   0x0A,
24   0x1F,
25   0x1F,
26   0x1F,
27   0x0E,
28   0x04,
29   0x00
30 };
31
```

```
30 //  
31 /**  
32 * @brief Delay function for producing delay in 0.1 ms increments  
33 * @param t milliseconds to be delayed  
34 * @return void  
35 */  
36 void delay(uint16_t t)  
37 {  
38     uint16_t i;  
39     for(i=t; i > 0; i--)  
40         __delay_cycles(100);  
41 }  
42  
43 /**  
44 * @brief Function to pulse EN pin after data is written  
45 * @return void  
46 */  
47 void pulseEN(void)  
48 {  
49     LCD_OUT |= EN;                      // Giving a falling edge at EN pin  
50     delay(1);  
51     LCD_OUT &= ~EN;  
52     delay(1);  
53 }  
54
```

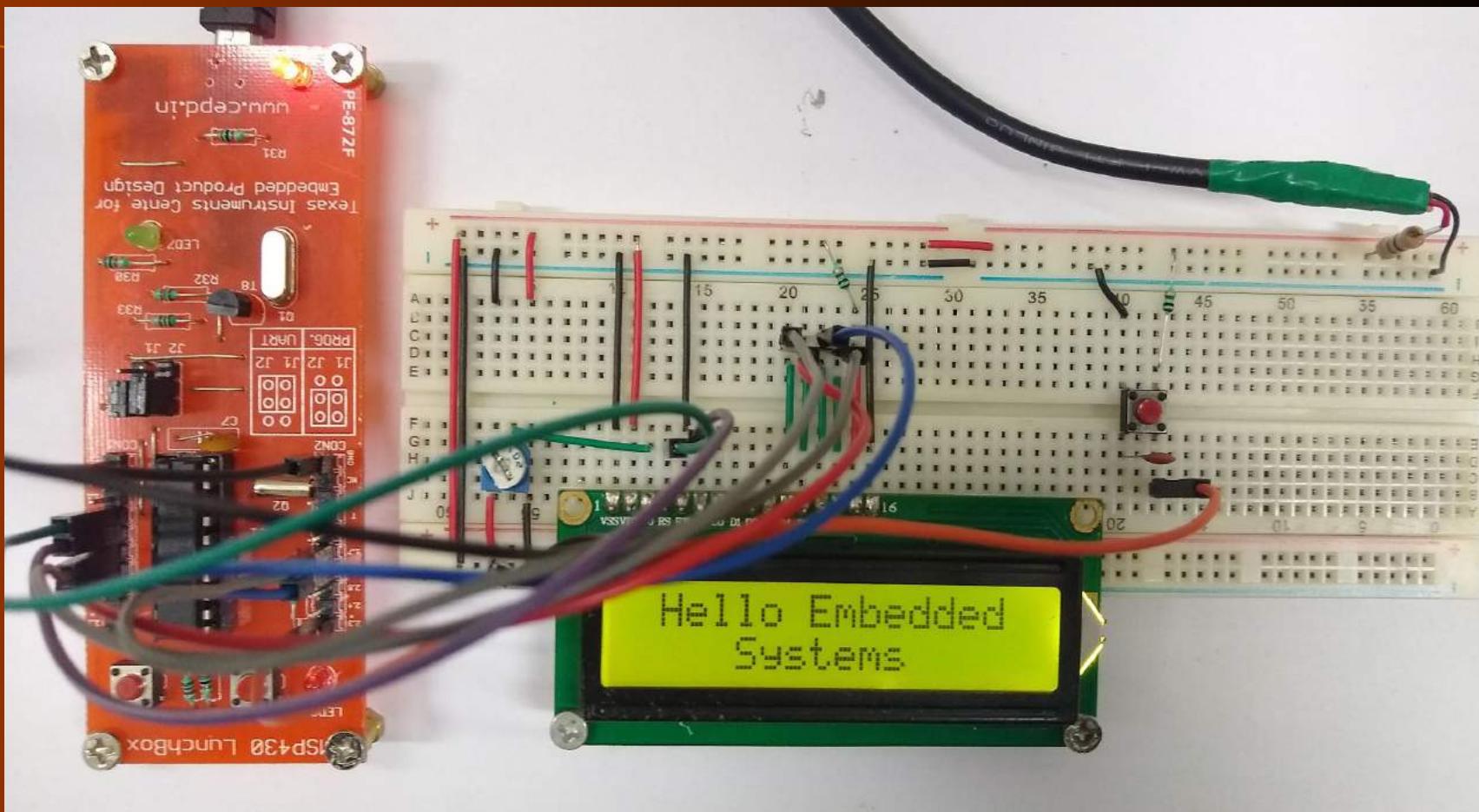
```
54
55 /**
56 * @brief Function to write data/command to LCD
57 * @param value Value to be written to LED
58 * @param mode Mode -> Command or Data
59 * @return void
60 */
61 void lcd_write(uint8_t value, uint8_t mode)
62 {
63     if(mode == CMD)
64         LCD_OUT &= ~RS;                      // Set RS -> LOW for Command mode
65     else
66         LCD_OUT |= RS;                     // Set RS -> HIGH for Data mode
67
68     LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));           // Write high nibble first
69     pulseEN();
70     delay(1);
71
72     LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));    // Write low nibble next
73     pulseEN();
74     delay(1);
75 }
```

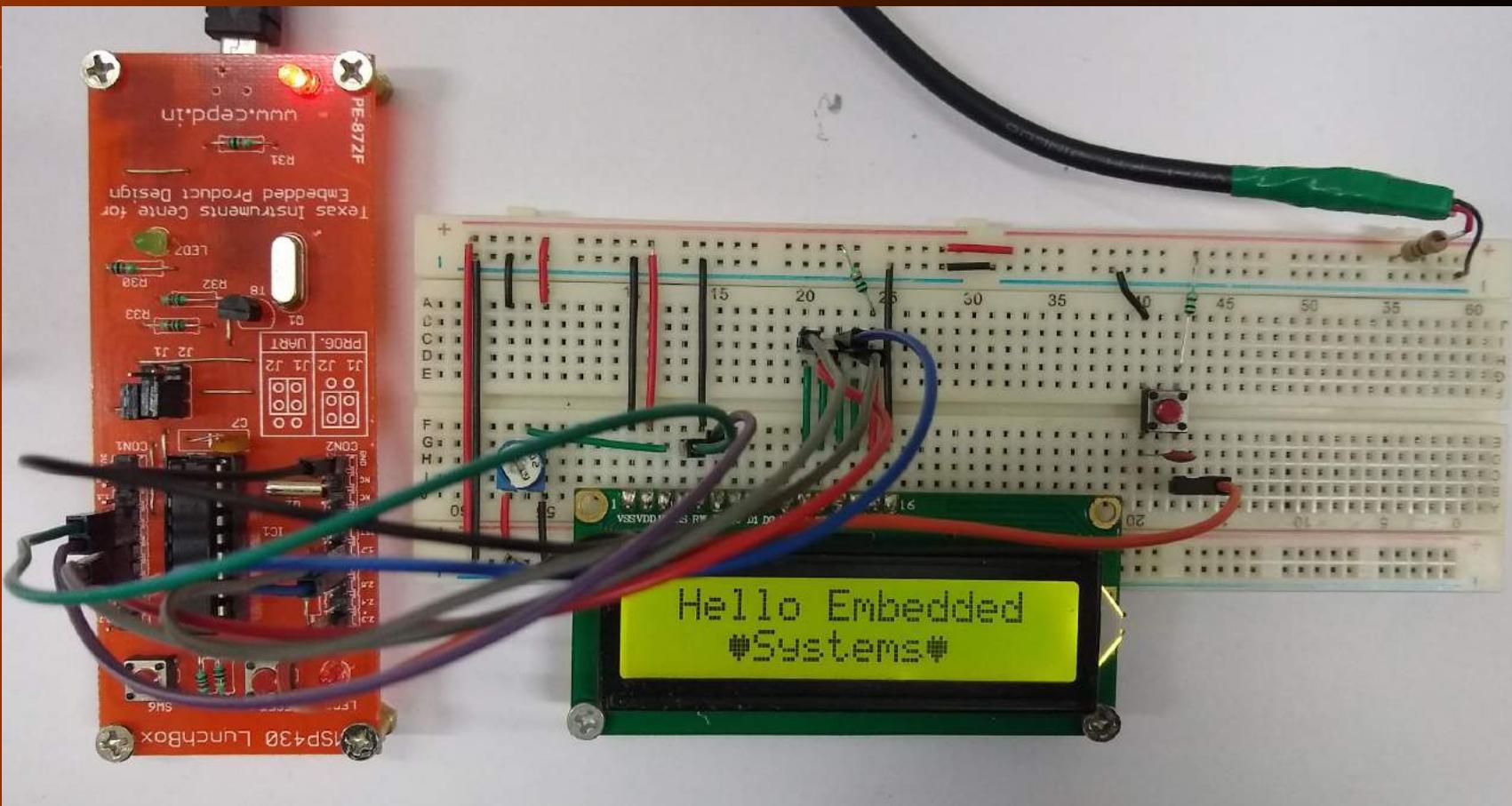
```
77 /**
78 * @brief Function to print a string on LCD
79 * @param s pointer to the character to be written.
80 * @return void
81 */
82 void lcd_print(char *s)
83 {
84     while(*s)
85     {
86         lcd_write(*s, DATA);
87         s++;
88     }
89 }
90
91 /**
92 * @brief Function to move cursor to desired position on LCD
93 * @param row Row Cursor of the LCD
94 * @param col Column Cursor of the LCD
95 * @return void
96 */
97 void lcd_setCursor(uint8_t row, uint8_t col)
98 {
99     const uint8_t row_offsets[] = { 0x00, 0x40};
100    lcd_write(0x80 | (col + row_offsets[row]), CMD);
101    delay(1);
102 }
103
```

```
110 void lcd_createChar(uint8_t location, uint8_t charmap[]) {
111     location &= 0x7; // we only have 8 locations 0-7
112     lcd_write(LCD_SETGRAMADDR | (location << 3), CMD);
113     int i = 0;
114     for (i=0; i<8; i++) {
115         lcd_write(charmap[i], DATA);
116     }
117 }
118
119 /**
120 *@brief Initialize LCD
121 */
122 void lcd_init()
123 {
124     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
125     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
126
127     delay(150);                      // Wait for power up ( 15ms )
128     lcd_write(0x33, CMD);            // Initialization Sequence 1
129     delay(50);                      // Wait ( 4.1 ms )
130     lcd_write(0x32, CMD);            // Initialization Sequence 2
131     delay(1);                       // Wait ( 100 us )
132
133     // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
134
135     lcd_write(0x28, CMD);           // 4 bit mode, 2 line
136     delay(1);
137
138     lcd_write(0x0C, CMD);           // Display ON, Cursor OFF, Blink OFF
139     delay(1);
140
141     lcd_write(0x06, CMD);           // Auto Increment Cursor
142     delay(1);
143
144     lcd_write(0x01, CMD);           // Clear screen
145     delay(20);
146
147     lcd_setCursor(0,0);             // Goto Row 1 Column 1
148 }
```

```
151
152 /*@brief entry point for the code*/
153 void main(void)
154 {
155     WDTCTL = WDTPW + WDTHOLD;          //! Stop Watchdog (Not recommended for code in production and devices working in field)
156
157     uint8_t count = 0;
158
159     P2DIR &=~ SW;
160
161     lcd_init();                      // Initialising LCD
162
163     lcd_createChar(0, heart);        // Creating Custom Character
164
165     lcd_setCursor(0,1);             // Cursor position (0,1)
166     lcd_print("Hello Embedded");    // Print
167
168     lcd_setCursor(1,4);             // Cursor position (1,3)
169     lcd_print("Systems");          // Print
170
```

```
171 while(1)
172 {
173     if(!(P2IN & SW))           // If SW is Pressed
174     {
175         __delay_cycles(20000); // Wait 20ms to debounce
176         while(!(P2IN & SW)); // Wait till SW Released
177         __delay_cycles(20000); // Wait 20ms to debounce
178         switch(count)
179         {
180             case 0:
181             {
182                 lcd_setCursor(1,3);                      // Cursor position (1,3)
183                 lcd_write(0x00, DATA);                  // Printing Custom Char (Heart)
184                 lcd_setCursor(1,11);                   // Cursor position (1,11)
185                 lcd_write(0x00, DATA);                  // Printing Custom Char [Heart]
186                 count = 1;
187                 break;
188             }
189
190             case 1:
191             {
192                 lcd_setCursor(1,3);                      // Cursor position (1,3)
193                 lcd_write(0x20, DATA);                  // Printing Space
194                 lcd_setCursor(1,11);                   // Cursor position (1,11)
195                 lcd_write(0x20, DATA);                  // Printing Space
196                 count = 0;
197                 break;
198             }
199         }
200     }
201 }
202 }
203 }
```







Thank you!

Introduction to Embedded System Design

MSP430 Timer Module

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

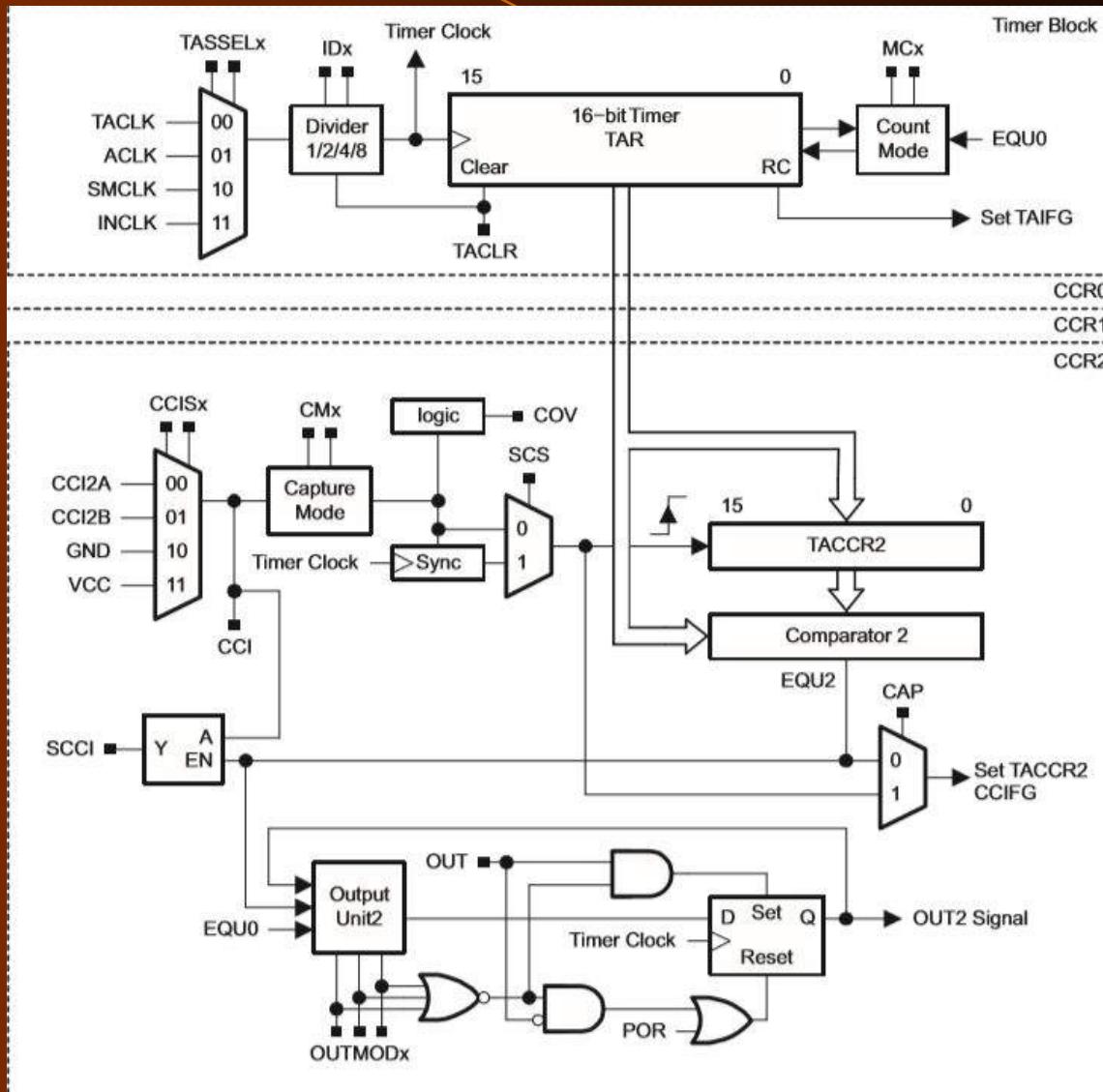
Why Timer?

- Timers are used for time-specific tasks like generating delays, baud rates or for counting events in fixed intervals of time.
- A dedicated hardware timer frees up the processor so that it can perform more non-time critical process.
- Software delay loops are a waste of the processor's capabilities. Also, delays written in C are not precise. So, a hardware timer peripheral inside a microcontroller helps!
- MSP430G2553 has 2 16-Bit Timer_A: Timer0_A and Timer1_A.

Features of TIMER_A In MSP430G2553

- Timer_A is a 16-bit counter with four operating modes. It can count till 65535 ($2^{16}-1$).
- Timer_A can be used for capture/compare and PWM signals.
- It can also be used to generate interrupts, which may be generated from the counter on overflow conditions or from any of the capture/compare registers.
- It can measure frequency or take time-stamp of the external inputs directly.
- Outputs can be driven at specified times precisely, either once or periodically.

TIMER_A Block Diagram



TIMER_A Major Components

- Timer_A counter register - TAR

- This is a 16-bit register whose value increments or decrements with every rising edge of clock signal.
- Behaviour (increment or decrement) of TAR register can be selected by configuring the mode of the timer.

15	14	13	12	11	10	9	8
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
TARx	Bits 15-0	Timer_A register. The TAR register is the count of Timer_A.					

TIMER_A Major Components

- Timer_A capture/compare register (TACCRx)
 - There are three capture/compare registers in each timer:
 - TA0CCR0, TA0CCR1, TA0CCR2 for Timer0_A
 - TA1CCR0, TA1CCR1, TA1CCR2 for Timer1_A.
 - All the three channels share the same TAR.

15	14	13	12	11	10	9	8
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
TACCRx	Bits 15-0	Timer_A capture/compare register. Compare mode: TACCRx holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TACCRx register when a capture is performed.					

Capture vs Compare Mode

- In capture mode, the Timer value (TAR) is captured in this TACCRx register in an external or internal input event.
- In compare mode, Timer value (TAR) is compared with this TACCRx register and operations are performed based on the compare event.
- The compare mode is used to generate PWM output signals or interrupts at specific time intervals.

Clock Source for Timer_A

- The timer clock can be sourced from ACLK, SMCLK, or externally via TAACLK or INCLK (For Timer1_A in case of MSP430G2553, TAACLK and INCLK is not present).
- The clock source is selected with the TASSELx bits.
- The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the IDx bits.
- The timer clock divider is reset when TACLR is set.

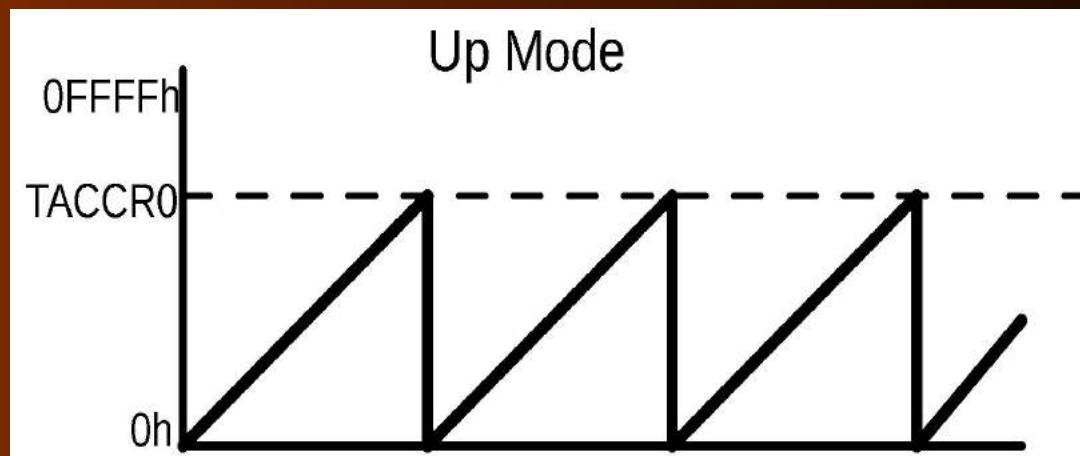
Timer Modes

Timer's operation mode can be selected by configuring 'MCx' bits in the Timer Control TACTL register

MC	Mode	Description
00	Stop	Timer is stopped.
01	Up	Timer repeatedly counts from zero to value stored in TACCR0 Register
10	Continuous	Timer repeatedly counts from zero to 0xFFFF hex.
11	Up/Down	Timer repeatedly counts from zero to value stored in TACCR0 and then back to zero.

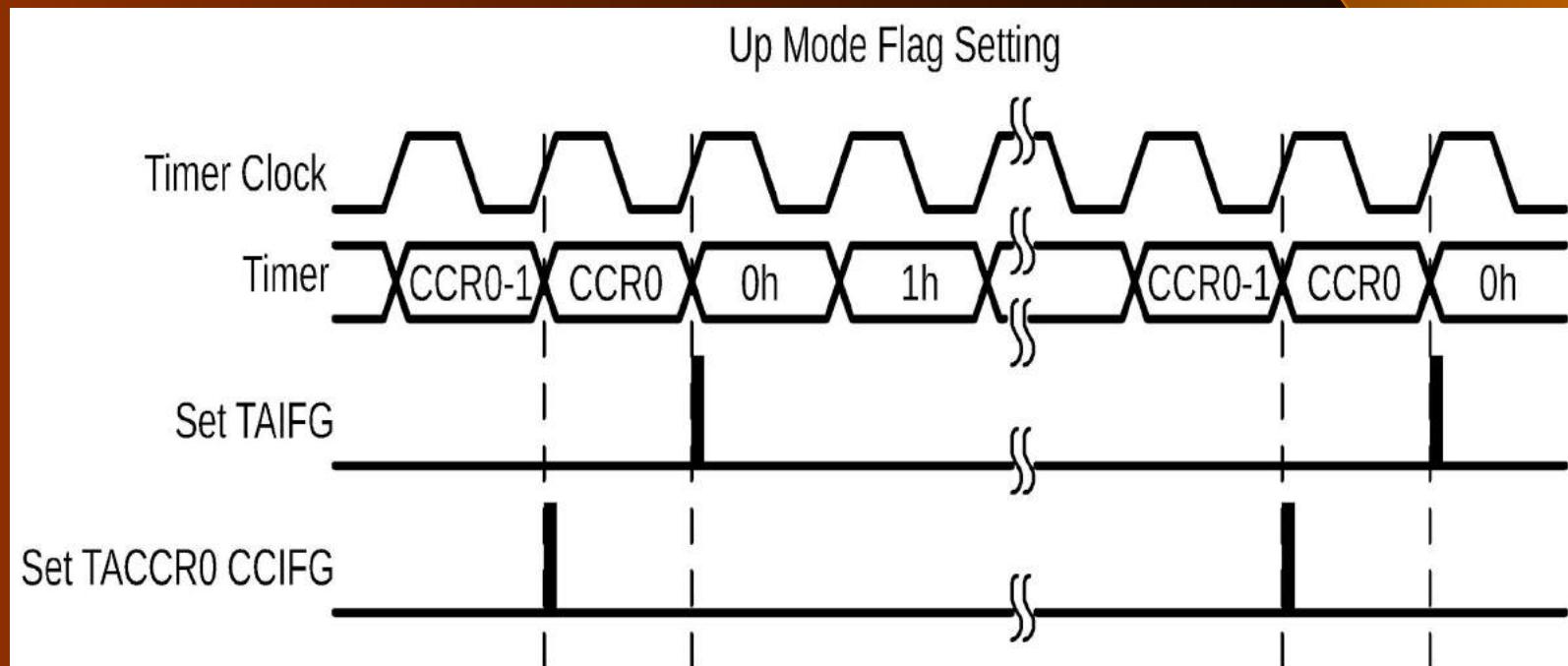
Up Mode

- The timer repeatedly counts up to the value of compare register TACCR0 starting from 0000h.
- The number of timer counts in the period is $TACCR0+1$. When the timer value equals TACCR0 the timer restarts counting from zero.
- The up mode is used if the timer period is required to be different from 0FFFFh counts..



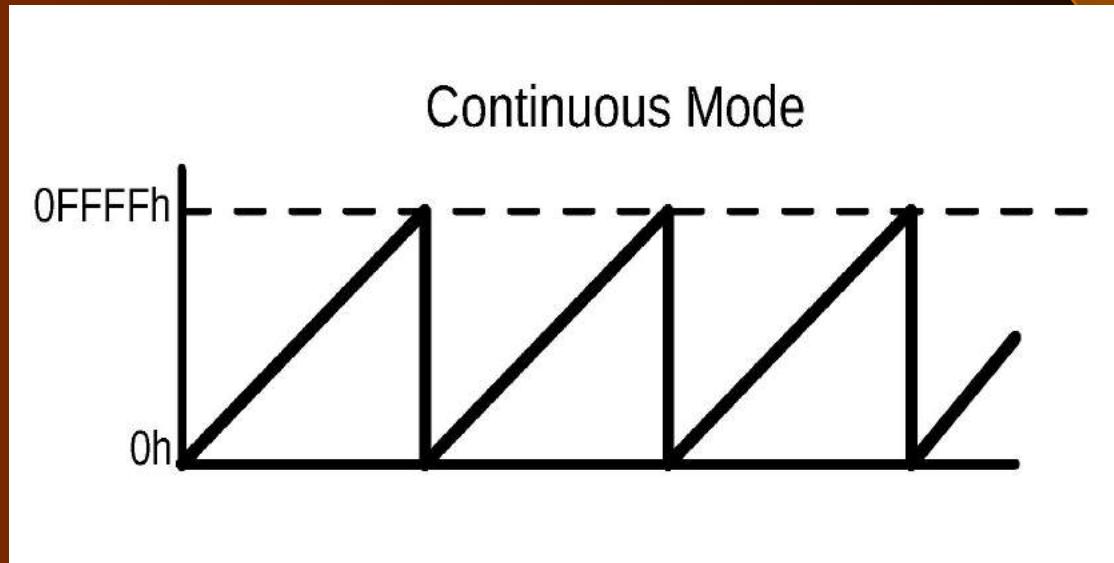
Timer Interrupts in Up Mode

- The TACCR0 CCIFG interrupt flag is set when the timer *counts* to the TACCR0 value.
- The TAIFG interrupt flag is set when the timer *counts* from TACCR0 to zero.



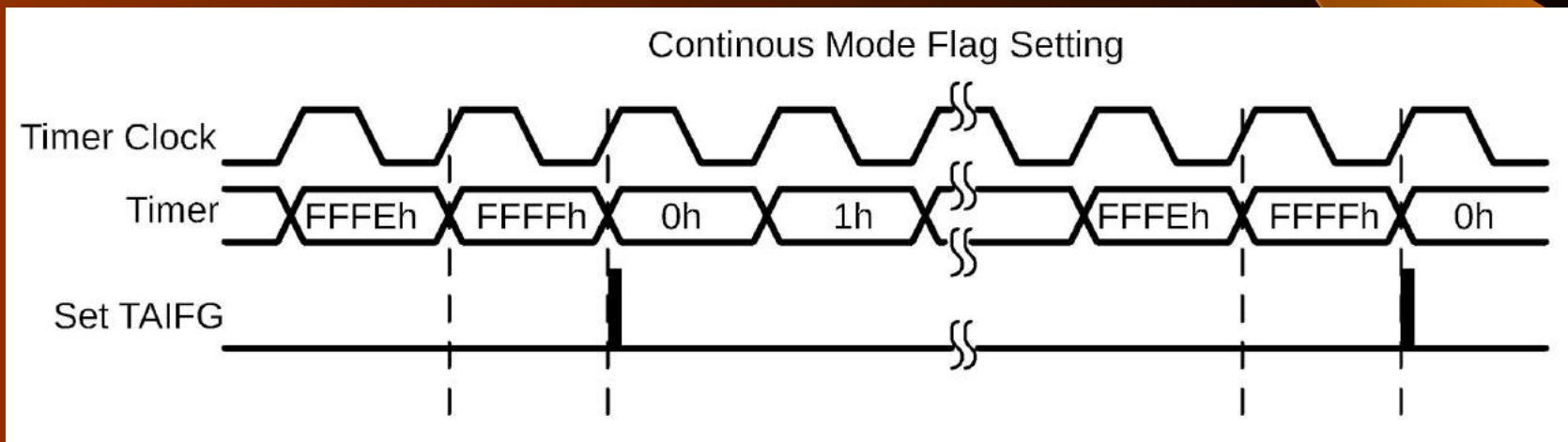
Continuous Mode

- In the continuous mode, the timer repeatedly counts up to FFFFh and restarts from zero.



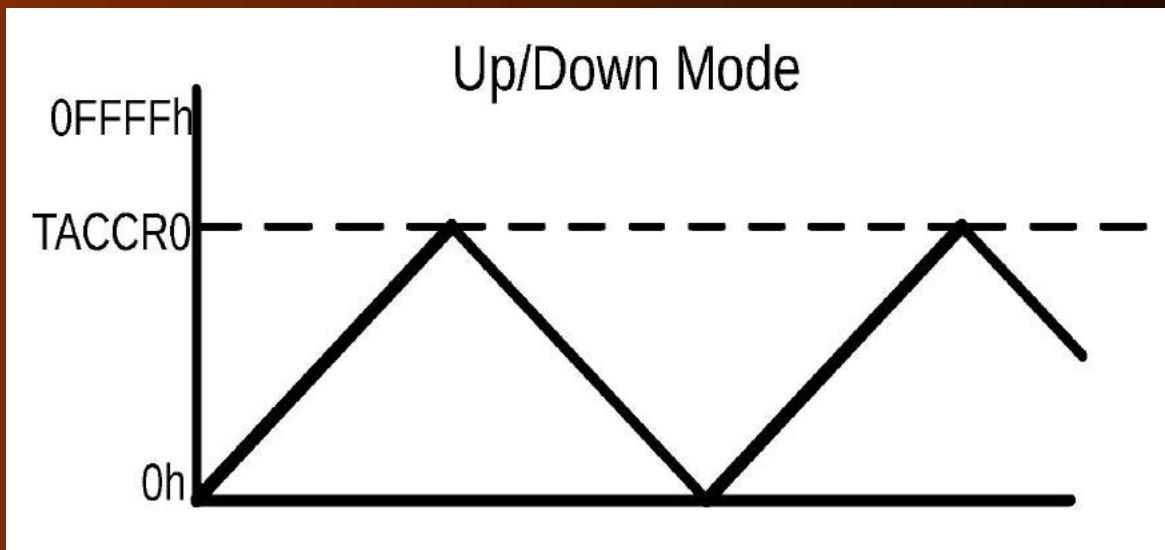
Timer Interrupts in Continuous Mode

- The TAIFG interrupt flag is set when the timer *counts* from FFFFh to zero.



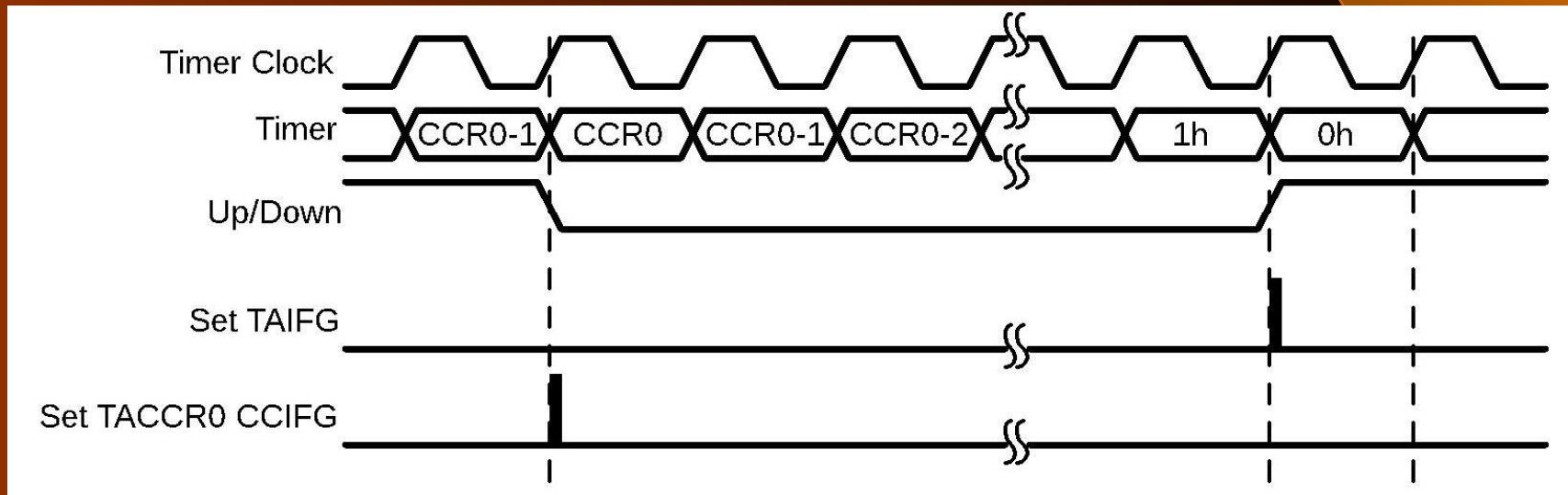
Up/Down Mode

- The timer repeatedly counts up to the value of compare register TACCR0 and back down to zero.
- The up/down mode is used if the timer period must be different from 0FFFFh counts, and if a symmetrical pulse generation is needed.



Timer Interrupts in Up/Down Mode

- The TACCR0 CCIFG interrupt flag is set when the timer *counts* from TACCR0 – 1 to TACCR0
- TAIFG is set when the timer completes counting down from 0001h to 0000h.



TIMER_A Registers

1. Timer_A control - TACTL
2. Timer_A counter - TAR
3. Timer_A capture/compare control 0 - TACCTL0
4. Timer_A capture/compare 0 - TACCR0
5. Timer_A capture/compare control 1 - TACCTL1
6. Timer_A capture/compare 1 - TACCR1
7. Timer_A capture/compare control 2 - TACCTL2
8. Timer_A capture/compare 2 - TACCR2
9. Timer_A Interrupt vector - TAIV

TACTL, Timer_A Control Register

TACCTL Continued

TASSELx	Bits 9-8	Timer_A clock source select
	00	TACLK
	01	ACLK
	10	SMCLK
	11	INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock.
	00	/1
	01	/2
	10	/4
	11	/8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.
	00	Stop mode: the timer is halted.
	01	Up mode: the timer counts up to TACCR0.
	10	Continuous mode: the timer counts up to 0FFFFh.
	11	Up/down mode: the timer counts up to TACCR0 then down to 0000h.
Unused	Bit 3	Unused
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request.
	0	Interrupt disabled
	1	Interrupt enabled
TAIFG	Bit 0	Timer_A interrupt flag
	0	No interrupt pending
	1	Interrupt pending

TACCTLx, Capture/Compare Control Register

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx		CCIE		CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

Timer_A has 3 channels each being controlled by TACCTLx. Each channel has its TACCRx register.

TACCTLx Continued

CMx	Bit 15-14	Capture mode
	00	No capture
	01	Capture on rising edge
	10	Capture on falling edge
	11	Capture on both rising and falling edges
CCISx	Bit 13-12	Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections.
	00	CCIxA
	01	CCIxB
	10	GND
	11	V _{cc}
SCS	Bit 11	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
	0	Asynchronous capture
	1	Synchronous capture
SCCI	Bit 10	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit
Unused	Bit 9	Unused. Read only. Always read as 0.
CAP	Bit 8	Capture mode
	0	Compare mode
	1	Capture mode

TACCTLx Continued

OUTMODx	Bits 7-5	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0.
	000	OUT bit value
	001	Set
	010	Toggle/reset
	011	Set/reset
	100	Toggle
	101	Reset
	110	Toggle/set
	111	Reset/set
CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.
	0	Interrupt disabled
	1	Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
OUT	Bit 2	Output. For output mode 0, this bit directly controls the state of the output.
	0	Output low
	1	Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
	0	No capture overflow occurred
	1	Capture overflow occurred
CCIFG	Bit 0	Capture/compare interrupt flag
	0	No interrupt pending
	1	Interrupt pending

TAIV, Timer_A Interrupt Vector Register

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	0	0	0		TAIVx		0
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TAIVx Bits 15-0 Timer_A interrupt vector value

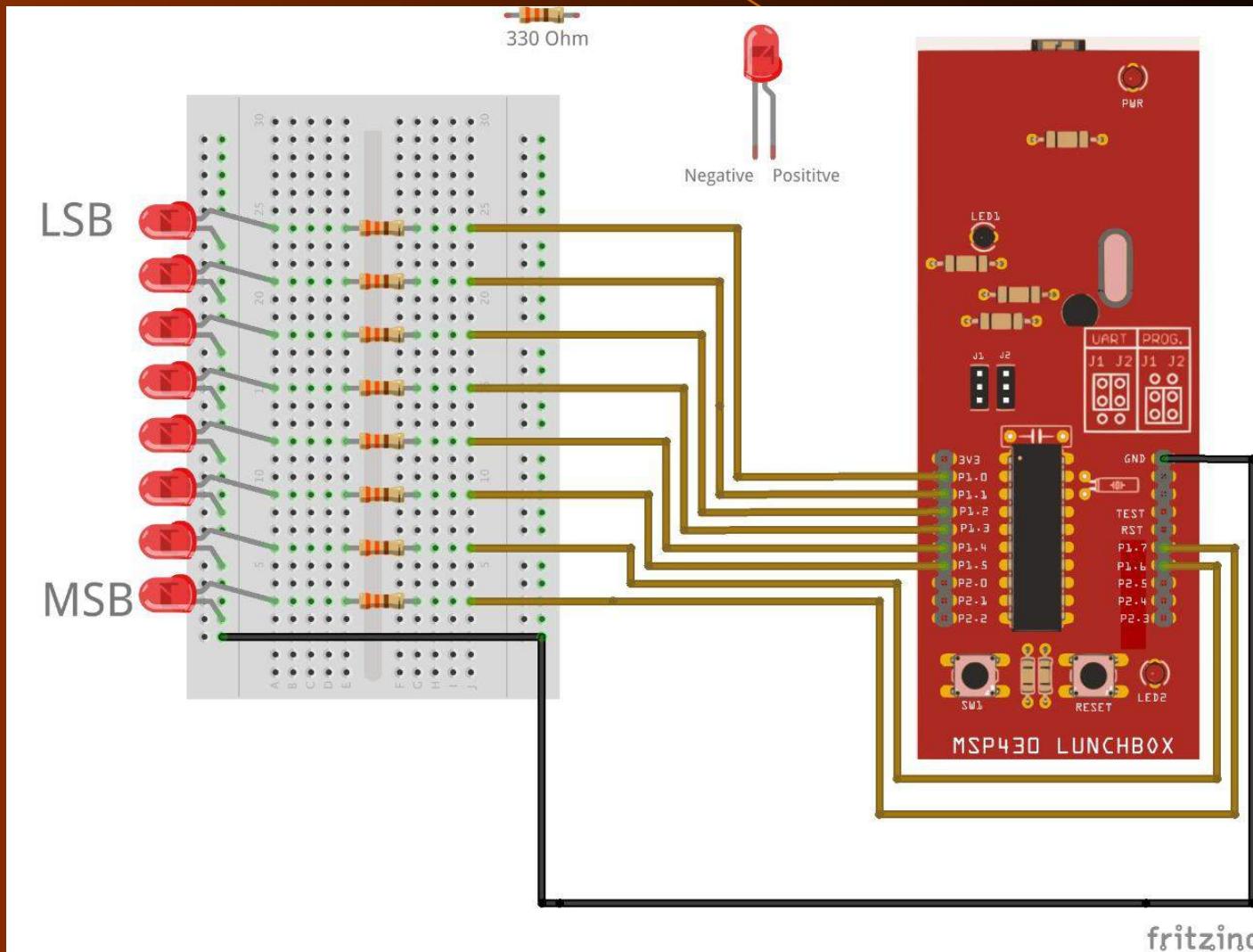
TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	-	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2 ⁽¹⁾	TACCR2 CCIFG	
06h	Reserved	-	
08h	Reserved	-	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	-	
0Eh	Reserved	-	Lowest

⁽¹⁾ Not implemented in MSP430x20xx devices

TIMER_A Interrupts

- Timer Overflow sets the flag TAIFG in TACTL.
- Interrupt vectors associated with the 16-bit Timer_A module:
 - TACCR0 interrupt vector for TACCR0 CCIFG.
 - TAIV interrupt vector for all other CCIFG flags and TAIFG.
- In capture mode any CCIFG flag is set when a timer value is captured in the associated TACCRx register.
- In compare mode, any CCIFG flag is set if TAR counts to the associated TACCRx value.
- Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

Circuit Diagram



Example Code : Hello_Timer

```
1 #include <msp430.h>
2 #include <inttypes.h>
3
4 volatile char count = 0;
5 volatile unsigned int i;
6
7 /**
8 * @brief
9 * These settings are wrt enabling GPIO on Lunchbox
10 */
11 void register_settings_for_GPIO()
12 {
13     P1DIR |= 0xFF;                      //P1.0 to P1.7 are set as Output
14     P1OUT |= 0x00;                      //Initially they are set to logic zero
15 }
16
17 /**
18 * @brief
19 * These settings are w.r.t enabling TIMER0 on Lunch Box
20 */
21 void register_settings_for_TIMER0()
22 {
23     CCTL0 = CCIE;                      // CCR0 interrupt enabled
24     TACTL = TASSEL_1 + MC_1;           // ACLK = 32768 Hz, upmode
25     CCR0 = 32768;                     // 1 Hz
26 }
```

Example Code :

Hello_Timer

```
27
28 /*@brief entry point for the code*/
29 void main(void)
30 {
31     WDTCTL = WDTPW + WDTHOLD;           //! Stop Watch dog (Not recommended for code in production and devices working in field)
32
33     do{
34         IFG1 &= ~OFIFG;                // Clear oscillator fault flag
35         for (i = 50000; i--;);        // Delay
36     } while (IFG1 & OFIFG);          // Test osc fault flag
37
38     register_settings_for_TIMER0();
39     register_settings_for_GPIO();
40     _BIS_SR(LPM3_bits + GIE);       // Enter LPM3 w/ interrupt
41 }
42
43 /*@brief entry point for TIMER0 interrupt vector*/
44 #pragma vector= TIMER0_A0_VECTOR
45 __interrupt void Timer_A (void)
46 {
47     count++;
48     P1OUT = count;                 //Assign value of Count to PORT 1 to represent it as binary number
49 }
```



Thank you!

Introduction to Embedded System Design

Pulse Width Modulation Timer Compare: Pulse Width Modulation

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

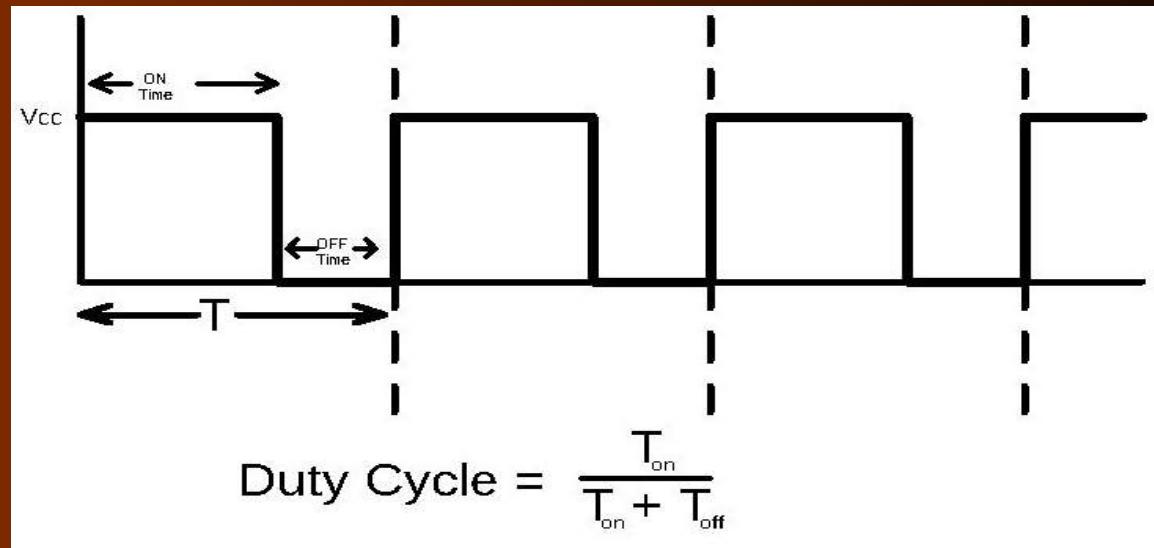
Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Pulse Width Modulation

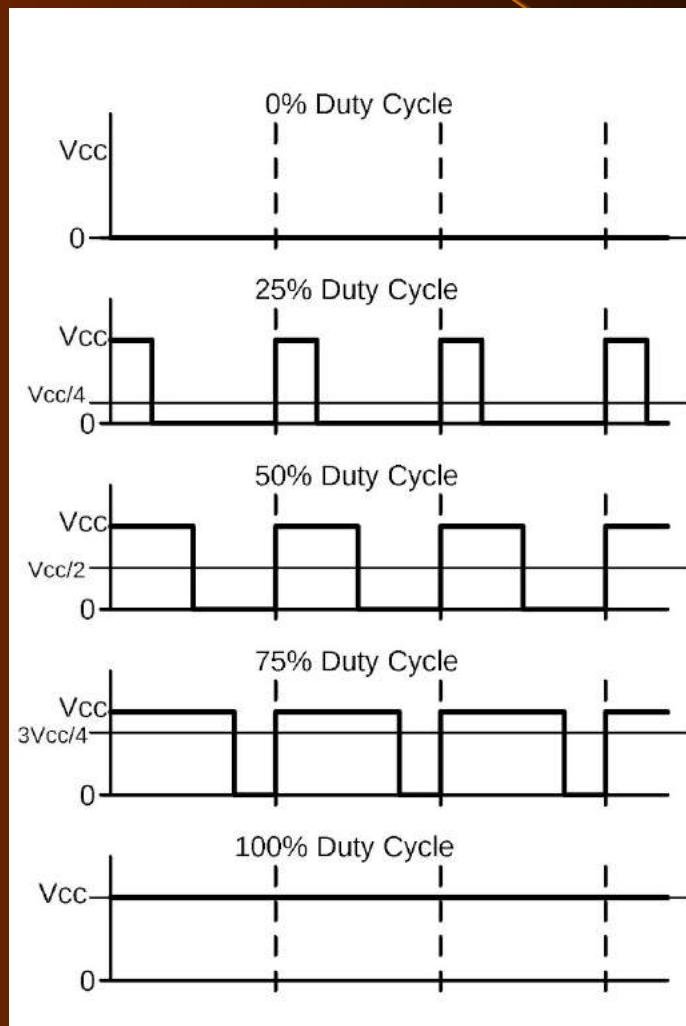
- PWM is a technique in which the width of a pulse is modulated to vary the average value of the signal.
- The width refers to the ON-time of the Rectangular pulse. The time period of the signal is kept fixed.



Pulse Width Modulation

- As the On-Time increases, the Duty Cycle of the Pulse increases and the average value of the signal increases.
- If a PWM signal is applied to an LED such that it is turned on and off at a very high rate that the human eye cannot detect the changes due to persistence of vision, the intensity of the led seems to vary.
- The output device operates on average value of this pulse.
- This mimics analog voltages through a digital output.

Average Value of Signals of Different Duty Cycles



Software PWM

- Software PWM is implemented with the help of delays.

Step 1: Turn the Output Pin On using software by using PxOUT

Step 2: Give some delay (Ton)

Step 3: Turn the Output Pin Off using software by using PxOUT

Step 4: Give some delay (Toff)

- Software PWM is helpful when the pin cannot be configured as a Timer Output Pin.
- PWM Time Period = Ton + Toff.

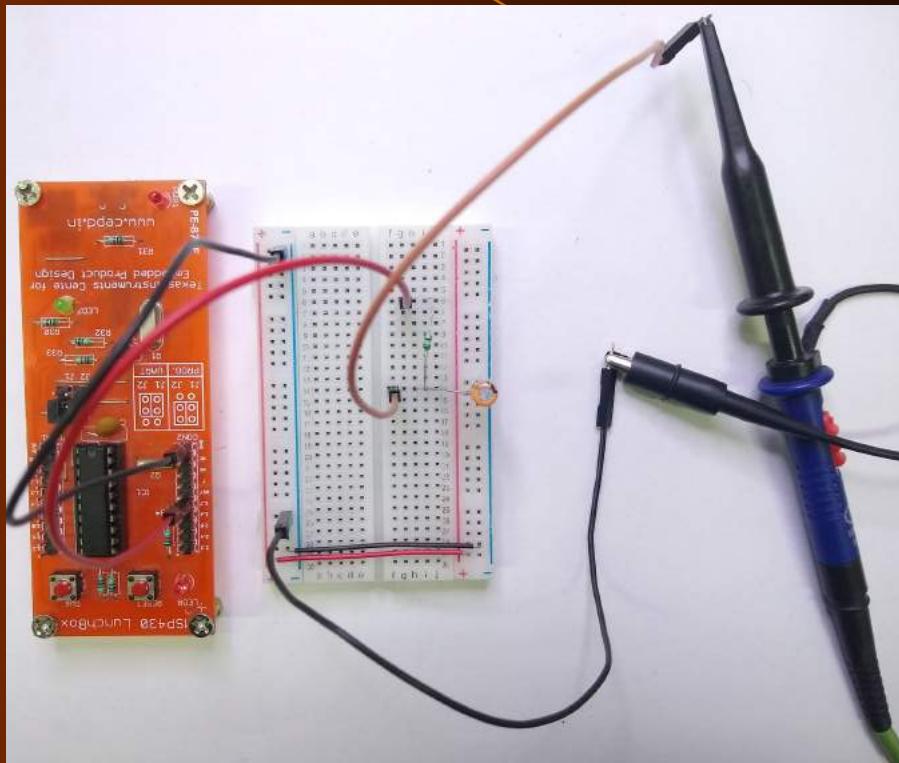
Software PWM Code with Large Delay

```
1 #include <msp430.h>
2
3 #define RED    BIT7           // Red LED -> P1.7
4
5 /**
6 * @brief This function provides delay
7 * @param unsigned int
8 * @return void
9 */
10 void delay(unsigned int t)      // Custom delay function
11 {
12     unsigned int i;
13     for(i = t; i > 0; i--)
14     {
15         __delay_cycles(50);          // __delay_cycles accepts only constants !
16     }
17 }
18
19 /*@brief entry point for the code*/
20 void main(void)
21 {
22     unsigned int j;
23     WDTCTL = WDTPW | WDTHOLD;      // Stop watchdog timer
24
25     P1DIR |= RED;                // Set LED pin -> Output
26
27     while(1)
28     {
29
30         for(j = 0; j < 256; j++).....// Increasing Intensity
31         {
32             P1OUT |= RED;           // LED ON
33             if (j !=0) delay(j);   // Delay for ON Time
34             P1OUT &= ~RED;         // LED OFF
35             if ((255-j)!=0) delay(255-j); // OFF Time = Period - ON Time
36         }
37         for(j = 255; j > 0; j--).....// Decreasing Intensity
38         {
39             P1OUT |= RED;           // LED ON
40             if (j !=0) delay(j);   // Delay for ON Time
41             P1OUT &= ~RED;         // LED OFF
42             if ((255-j) != 0 ) delay(255-j); // OFF Time = Period - ON Time
43         }
44 }
```

Software PWM Code with Less Delay

```
1 #include <msp430.h>
2
3 #define RED    BIT7          // Red LED -> P1.7
4
5 /**
6 * @brief This function provides delay
7 * @param unsigned int
8 * @return void
9 */
10 void delay(unsigned int t)      // Custom delay function
11 {
12     unsigned int i;
13     for(i = t; i > 0; i--)
14     {   __delay_cycles(1);           // __delay_cycles accepts only constants !
15     }
16 }
17
18 /*@brief entry point for the code*/
19 void main(void)
20 {
21     unsigned int j;
22     WDTCTL = WDTPW | WDTHOLD;        // Stop watchdog timer
23
24     P1DIR |= RED;                  // Set LED pin -> Output
25
26     while(1)
27     {
28         for(j = 0; j < 256; j++).....// Increasing Intensity
29         {
30             P1OUT |= RED;           // LED ON
31             if (j !=0) delay(j);    // Delay for ON Time
32             P1OUT &= ~RED;          // LED OFF
33             if ((255-j)!=0) delay(255-j); // OFF Time = Period - ON Time
34         }
35         for(j = 255; j > 0; j--).....// Decreasing Intensity
36         {
37             P1OUT |= RED;           // LED ON
38             if (j !=0) delay(j);    // Delay for ON Time
39             P1OUT &= ~RED;          // LED OFF
40             if ((255-j) != 0 ) delay(255-j); // OFF Time = Period - ON Time
41         }
42     }
43 }
```

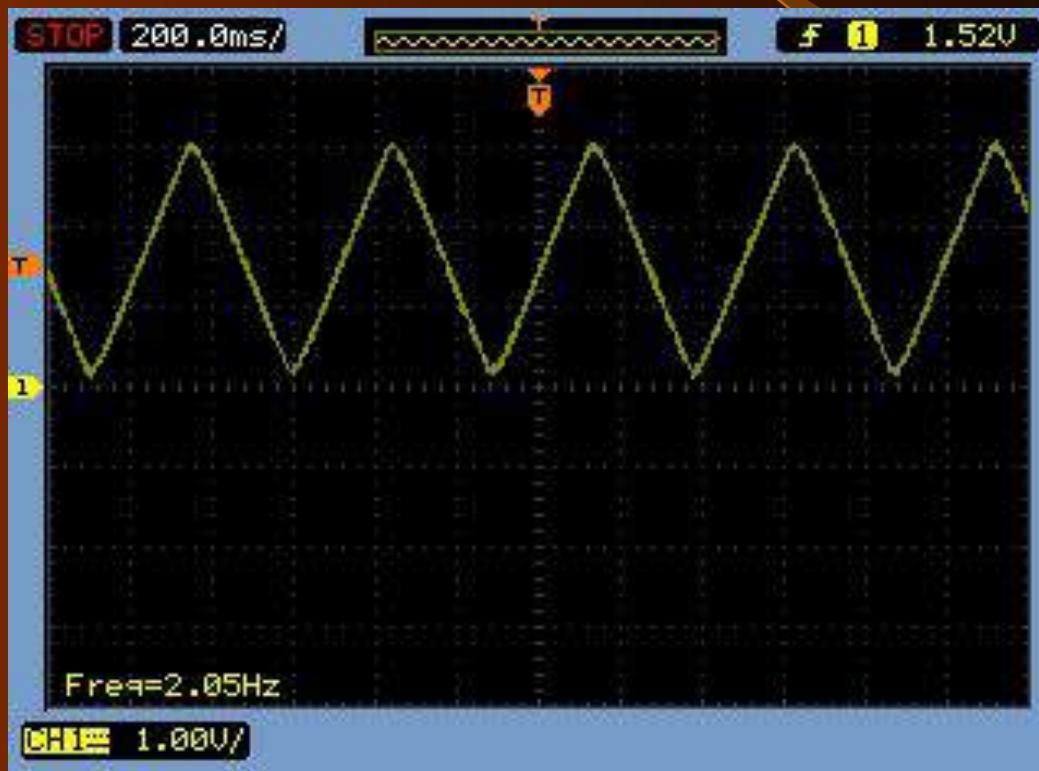
FILTER CONNECTION



$R = 1.5 \text{ Kohm}$, $C = 10 \mu\text{F}$

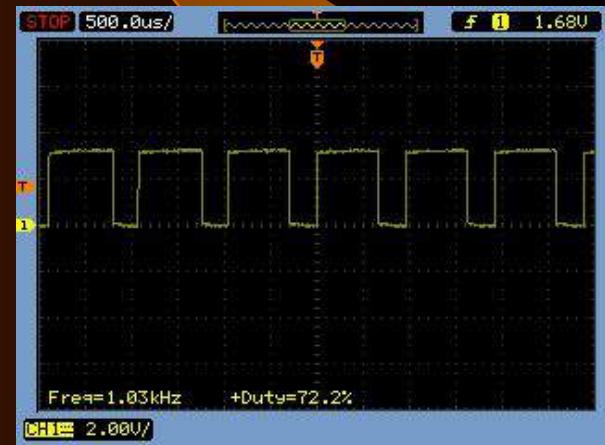
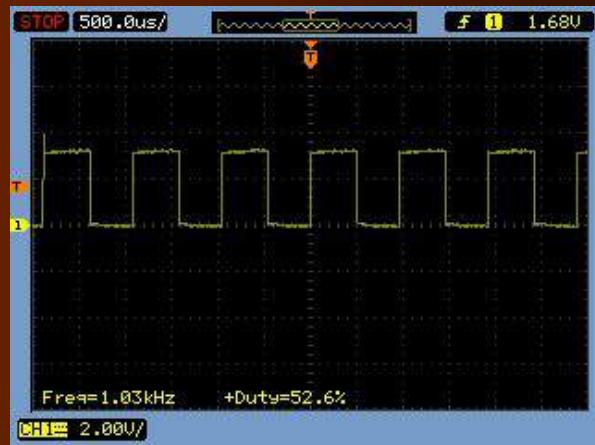
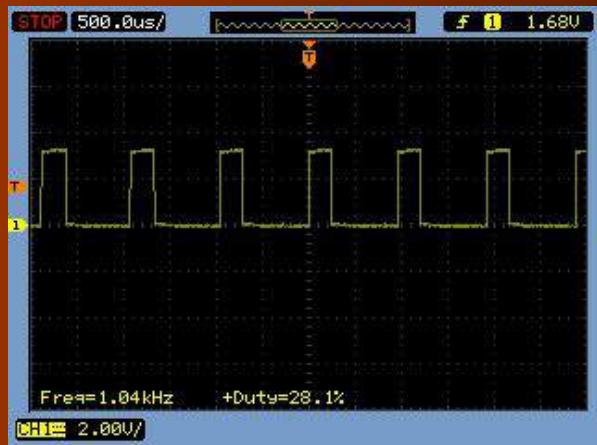
Filter Frequency = $1/(2*3.14*R*C) = 10.6 \text{ Hz}$

Filtered Output



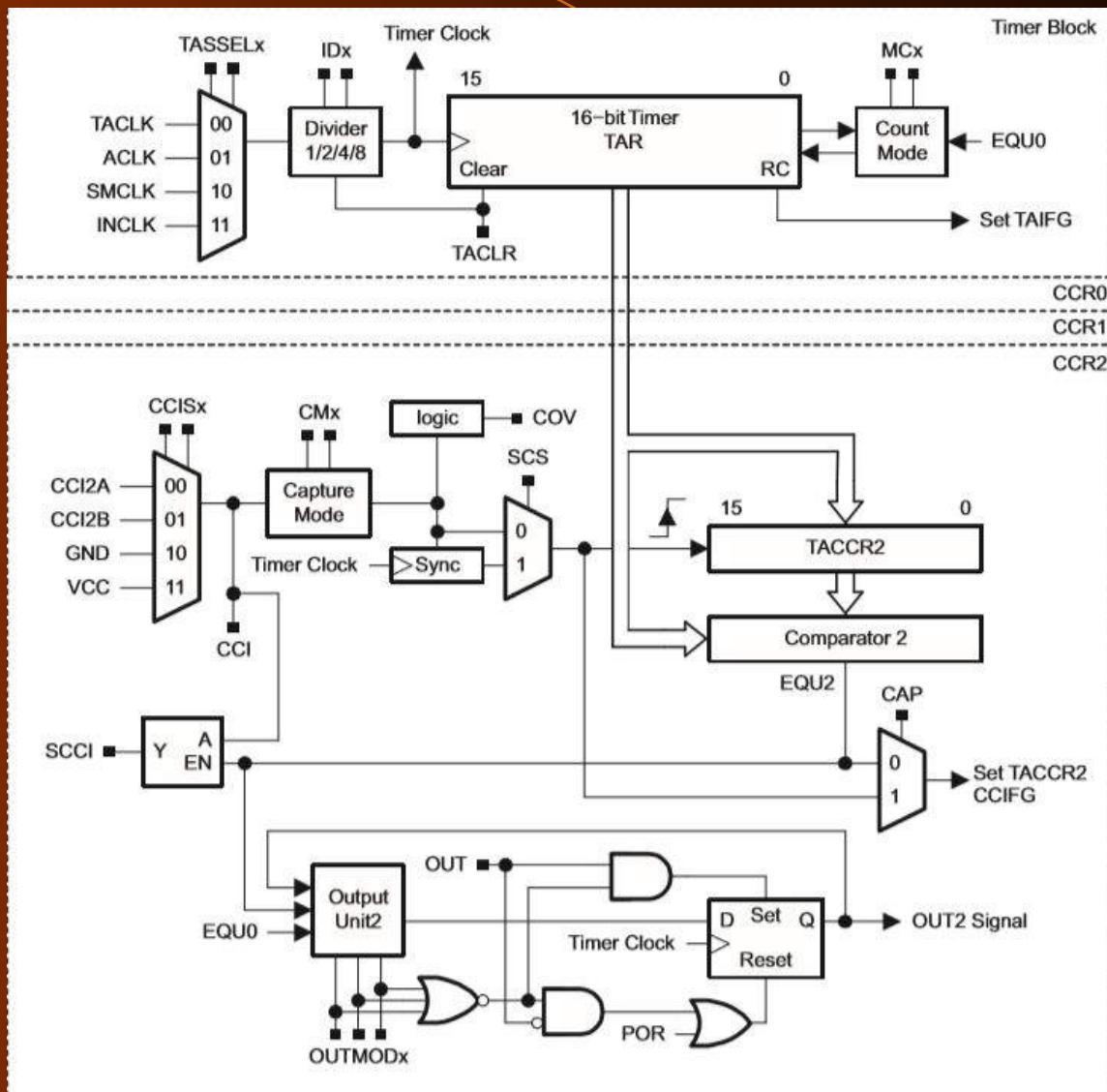
PWM signal at pin 1.7

Duty cycle of 28.1 %, 52.6 %, 72.2 %



Hardware PWM

TIMER_A Block Diagram



TIMER_A Output Unit

- Each capture/compare block contains an Output Unit.
- The compare mode is selected when bit CAP = 0 in TACCTLx register.
- When TAR value equals value in TACCRx, a internal signal EQUx is set, which affects the output according to output mode.
- The output unit is used to generate output signals such as PWM signals.
- Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals.
- The output modes are defined by the OUTMODn bits in TACCTLx register

Timer Modes

Timer's operation mode can be selected by configuring 'MCx' bits in the Timer Control TACTL register

MC	Mode	Description
00	Stop	Timer is stopped.
01	Up	Timer repeatedly counts from zero to value stored in TACCR0 Register
10	Continuous	Timer repeatedly counts from zero to 0xFFFF hex.
11	Up/Down	Timer repeatedly counts from zero to value stored in TACCR0 and then back to zero.

TIMER_A Output Modes

OUTMOD	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer <i>counts</i> to the TAxCRRn value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer <i>counts</i> to the TAxCRRn value. It is reset when the timer <i>counts</i> to the TAxCCR0 value.
011	Set/Reset	The output is set when the timer <i>counts</i> to the TAxCRRn value. It is reset when the timer <i>counts</i> to the TAxCCR0 value.
100	Toggle	The output is toggled when the timer <i>counts</i> to the TAxCRRn value. The output period is double the timer period.
101	Reset	The output is reset when the timer <i>counts</i> to the TAxCRRn value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer <i>counts</i> to the TAxCRRn value. It is set when the timer <i>counts</i> to the TAxCCR0 value.
111	Reset/Set	The output is reset when the timer <i>counts</i> to the TAxCRRn value. It is set when the timer <i>counts</i> to the TAxCCR0 value.

Output Mode Example with Up Mode

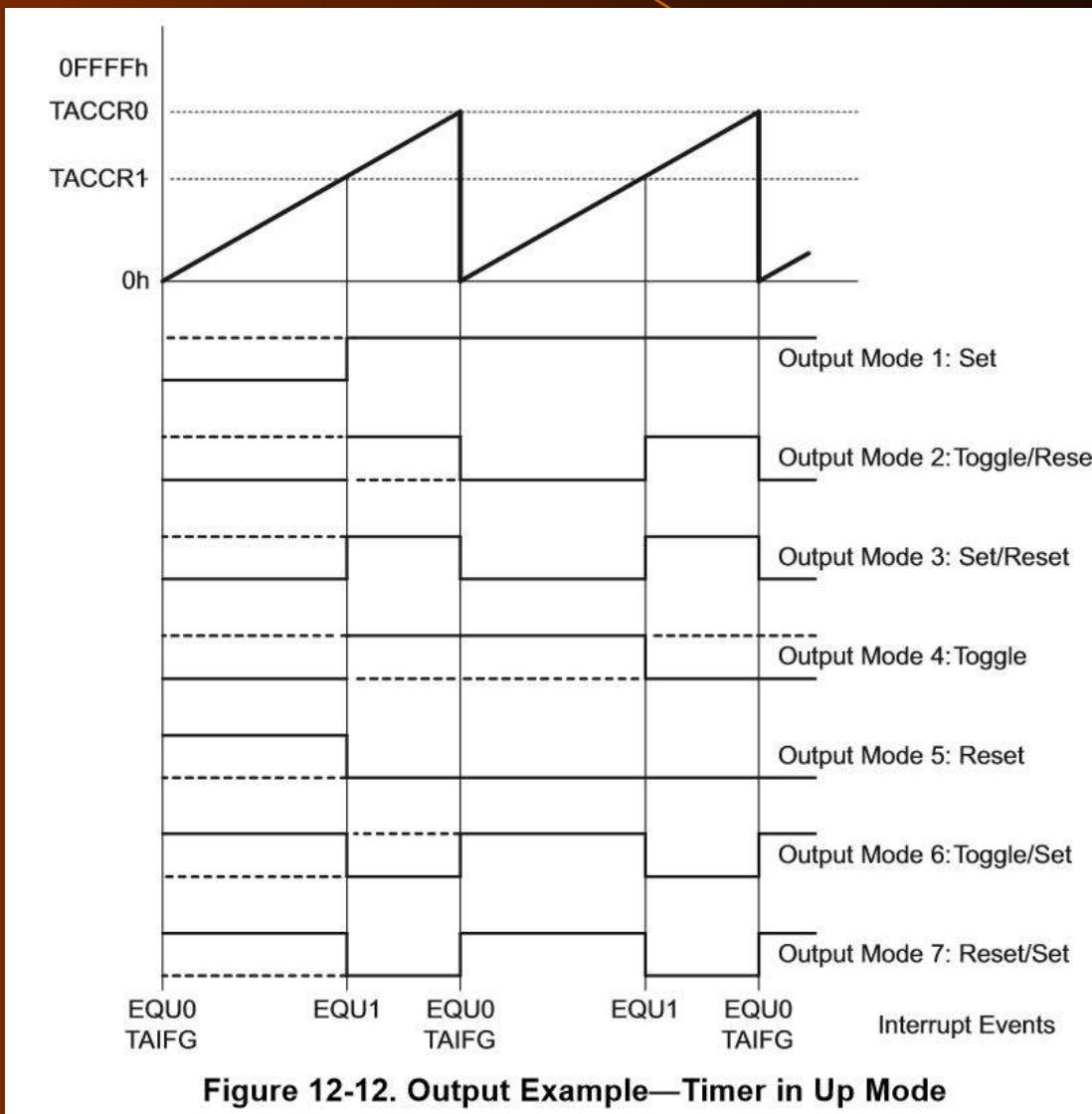
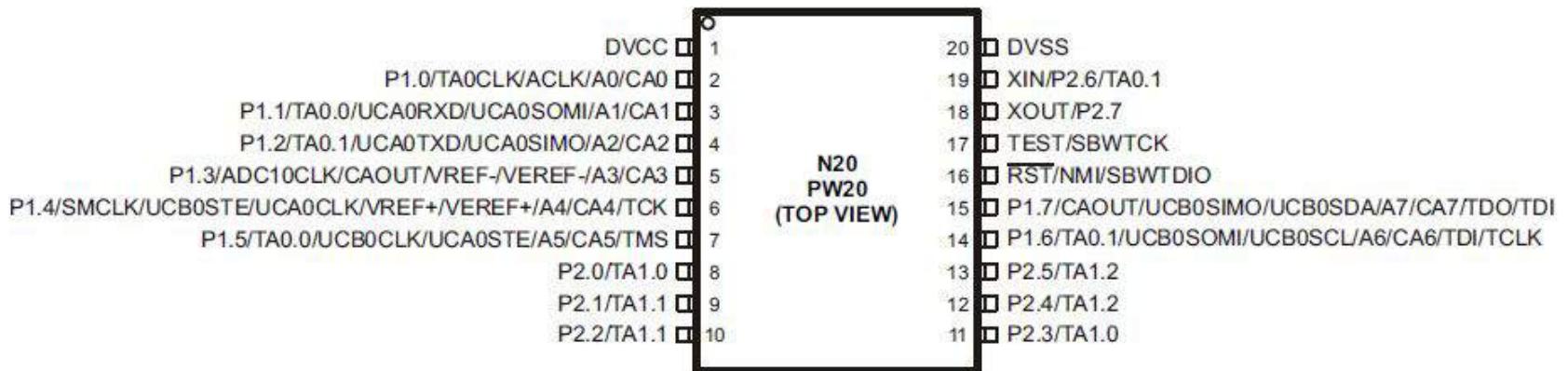


Figure 12-12. Output Example—Timer in Up Mode

PWM Using Timer

Step 1: To implement a Timer output, a pin which has timer output functionality has to be chosen. Let us choose P1.6 as TA0.1 output of Timer0.



NOTE: ADC10 is available on MSP430G2x53 devices only.

NOTE: The pulldown resistors of port P3 should be enabled by setting P3REN.x = 1.

MSP430G2553 Timer0_A Input and Output Pins

You can find this information on Page 16 of the datasheet:

INPUT PIN NUMBER			DEVICE INPUT SIGNAL	MODULE INPUT NAME	MODULE BLOCK	MODULE OUTPUT SIGNAL	OUTPUT PIN NUMBER		
PW20, N20	PW28	RHB32					PW20, N20	PW28	RHB32
P1.0-2	P1.0-2	P1.0-31	TACLK	TACLK	Timer	NA			
			ACLK	ACLK					
			SMCLK	SMCLK					
PinOsc	PinOsc	PinOsc	TACLK	INCLK					
P1.1-3	P1.1-3	P1.1-1	TA0.0	CCI0A					
			ACLK	CCI0B	CCR0	TA0	P1.1-3	P1.1-3	P1.1-1
			V _{ss}	GND			P1.5-7	P1.5-7	P1.5-5
			V _{cc}	V _{cc}				P3.4-15	P3.4-14
P1.2-4	P1.2-4	P1.2-2	TA0.1	CCI1A					
			CAOUT	CCI1B	CCR1	TA1	P1.2-4	P1.2-4	P1.2-2
			V _{ss}	GND			P1.6-14	P1.6-22	P1.6-21
			V _{cc}	V _{cc}			P2.6-19	P2.6-27	P2.6-26
	P3.0-9	P3.0-7	TA0.2	CCI2A				P3.5-19	P3.5-18
PinOsc	PinOsc	PinOsc	TA0.2	CCI2B	CCR2	TA2		P3.0-9	P3.0-7
			V _{ss}	GND				P3.6-20	P3.6-19
			V _{cc}	V _{cc}					

MSP430G2553 Timer1_A Input and Output Pins

You can find this information on Page 16 of the datasheet:

INPUT PIN NUMBER			DEVICE INPUT SIGNAL	MODULE INPUT NAME	MODULE BLOCK	MODULE OUTPUT SIGNAL	OUTPUT PIN NUMBER		
PW20, N20	PW28	RHB32					PW20, N20	PW28	RHB32
-	P3.7-21	P3.7-20	TACLK	TACLK	Timer	NA			
			ACLK	ACLK					
			SMCLK	SMCLK					
-	P3.7-21	P3.7-20	TACLK	INCLK					
P2.0-8	P2.0-10	P2.0-9	TA1.0	CCI0A			P2.0-8	P2.0-10	P2.0-9
P2.3-11	P2.3-16	P2.3-12	TA1.0	CCI0B	CCR0	TA0	P2.3-11	P2.3-16	P2.3-15
			V _{ss}	GND			P3.1-8	P3.1-6	
			V _{cc}	V _{cc}					
P2.1-9	P2.1-11	P2.1-10	TA1.1	CCI1A			P2.1-9	P2.1-11	P2.1-10
P2.2-10	P2.2-12	P2.2-11	TA1.1	CCI1B	CCR1	TA1	P2.2-10	P2.2-12	P2.2-11
			V _{ss}	GND			P3.2-13	P3.2-12	
			V _{cc}	V _{cc}					
P2.4-12	P2.4-17	P2.4-16	TA1.2	CCI2A	CCR2	TA2	P2.4-12	P2.4-17	P2.4-16
P2.5-13	P2.5-18	P2.5-17	TA1.2	CCI2B			P2.5-13	P2.5-18	P2.5-17
			V _{ss}	GND			P3.3-14	P3.3-13	
			V _{cc}	V _{cc}					

PWM Using Timer

- To select the timer function of that pin, PxDIR, PxSEL and PxSEL2 Registers are used.

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ⁽¹⁾					
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x=1 ⁽²⁾	JTAG Mode	CAPD.y
P1.5/ TA0.0/ UCB0CLK/ UCA0STE/ A5 ⁽²⁾ / CA5 TMS Pin Osc	5	P1.x (I/O)	I: 0; O: 1	0	0	0	0	0
		TA0.0	1	1	0	0	0	0
		UCB0CLK	from USCI	1	1	0	0	0
		UCA0STE	from USCI	1	1	0	0	0
		A5	X	X	X	1 (y = 5)	0	0
		CA5	X	X	X	0	0	1 (y = 5)
		TMS	X	X	X	0	1	0
		Capacitive sensing	X	0	1	0	0	0
P1.6/ TA0.1/ UCB0SOMI/ UCB0SCL/ A6 ⁽²⁾ / CA6 TDI/TCLK/ Pin Osc	6	P1.x (I/O)	I: 0; O: 1	0	0	0	0	0
		TA0.1	1	1	0	0	0	0
		UCB0SOMI	from USCI	1	1	0	0	0
		UCB0SCL	from USCI	1	1	0	0	0
		A6	X	X	X	1 (y = 6)	0	0
		CA6	X	X	X	0	0	1 (y = 6)
		TDI/TCLK	X	X	X	0	1	0
		Capacitive sensing	X	0	1	0	0	0

TIMER_A Registers

1. Timer_A control - TACTL
2. Timer_A counter - TAR
3. Timer_A capture/compare control 0 - TACCTL0
4. Timer_A capture/compare 0 - TACCR0
5. Timer_A capture/compare control 1 - TACCTL1
6. Timer_A capture/compare 1 - TACCR1
7. Timer_A capture/compare control 2 - TACCTL2
8. Timer_A capture/compare 2 - TACCR2
9. Timer_A Interrupt vector - TAIV

PWM Using Timer

Step 2: Timer is configured in Up mode, so TAR value goes from 0 to TACCR0 and repeats. Up mode is selected by ORing ‘MC_1’ to TACTL register.

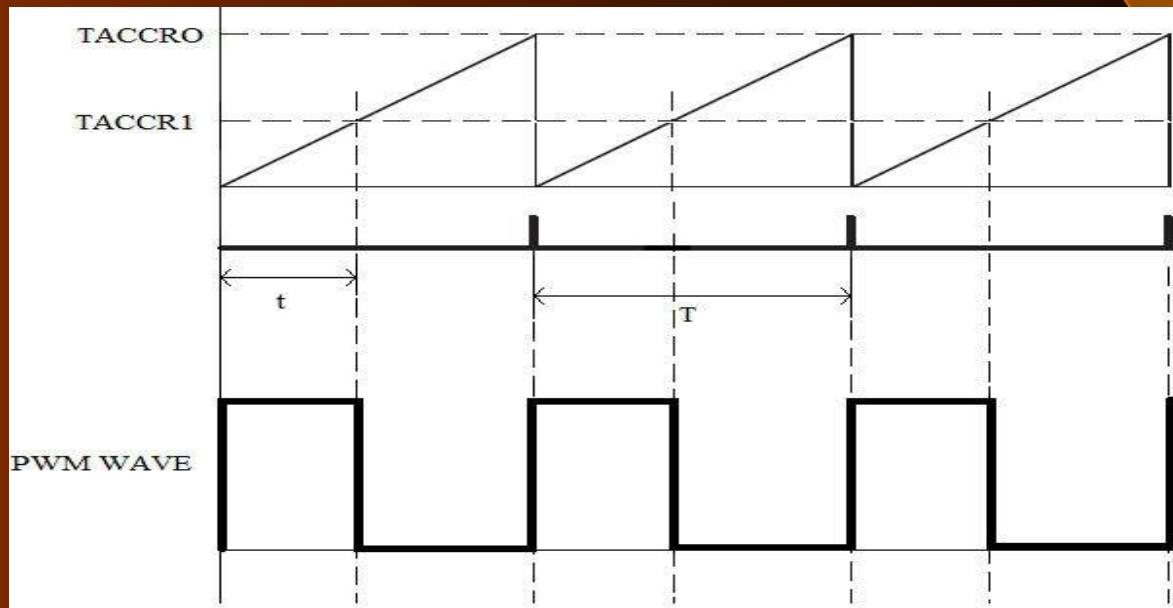
Step 3: The value of TACCR0 register is filled. TACCR0 sets time period (T) of PWM.

Step 4: The value of TACCR1 register is filled. TACCR1 sets the on-time (t) of PWM.

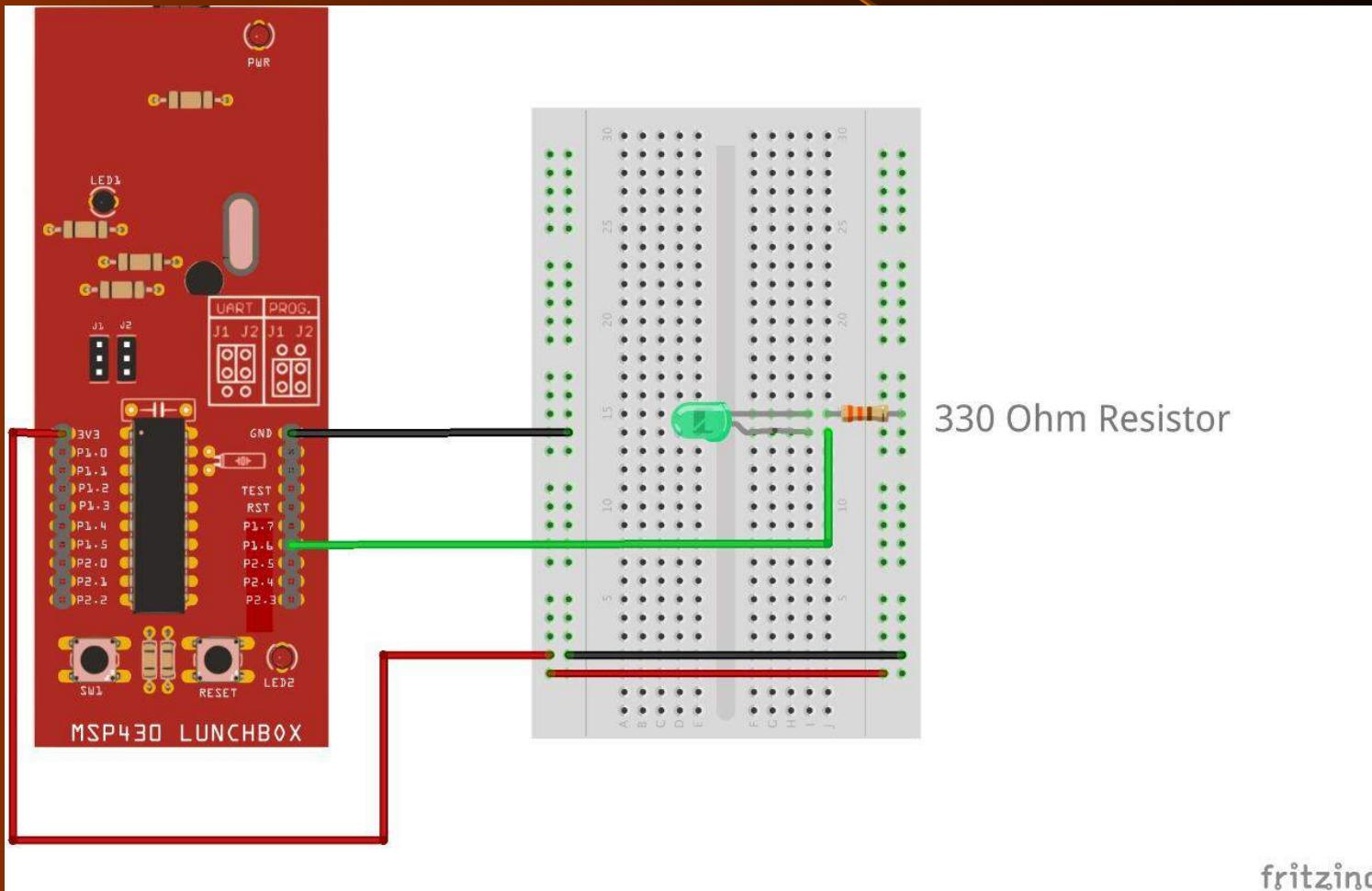
PWM Using Timer

Step 5: Output mode 7 is selected by writing ‘OUTMOD_7’ to TACCTL1 register : $TACCTL1 = OUTMOD_7;$

- Output resets when timer (TAR) value counts to TACCR1 value.
- Output value sets when timer (TAR) value count reaches TACCR0 value.



Circuit Diagram



Hardware PWM Code 1

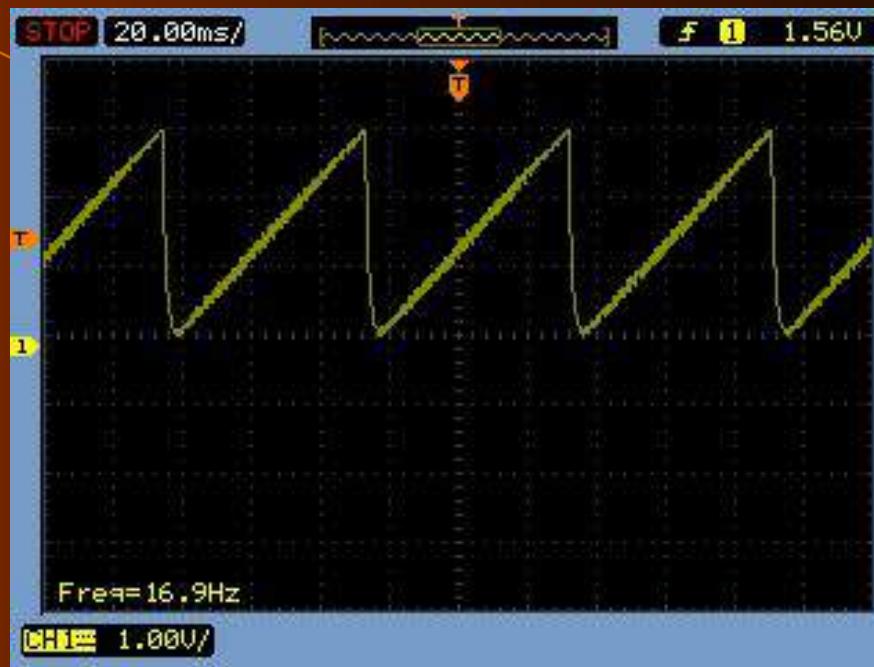
8 bit PWM at DCO freq -

1.1 MHz

```
1 #include <msp430.h>
2
3 #define GREEN BIT6                                // Green LED -> P1.6
4
5 /**
6 * @brief
7 * These settings are w.r.t enabling TIMER0 on Lunchbox
8 */
9 void register_settings_for_TIMER0()
10 {
11     P1DIR |= GREEN;                                // Green LED -> Output
12     P1SEL |= GREEN;                                // Green LED -> Select Timer Output
13
14     CCR0 = 255;                                    // Set Timer0 PWM Period
15     CCTL1 = OUTMOD_7;                             // Set TA0.1 Waveform Mode - Clear on Compare, Set on Overflow
16     CCR1 = 0;                                     // Set TA0.1 PWM duty cycle
17     CCTL0 = CCIE;                                 // CCR0 Enable Interrupt
18     TACTL = TASSEL_2 + MC_1;                      // Timer Clock -> SMCLK, Mode -> Up Count
19 }
20
```

```
20
21 /**
22 * @brief
23 * Entry point for the code
24 */
25 void main(void) {
26
27     WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer
28
29     register_settings_for_TIMER0();
30
31     __bis_SR_register(GIE);           // Enable CPU Interrupt
32
33     while(1)
34     {
35     }
36
37 }
38
39 /**
40 * @brief
41 * Entry point for TIMER0 interrupt vector
42 */
43 #pragma vector = TIMER0_A0_VECTOR
44 _interrupt void Timer_A(void)
45 {
46     CCR1 = CCR1 + 1;                 // Increment CCR1
47     if(CCR1 == 256)
48     {
49         CCR1 = 0;
50     }
51 }
52
```

Filtered Output

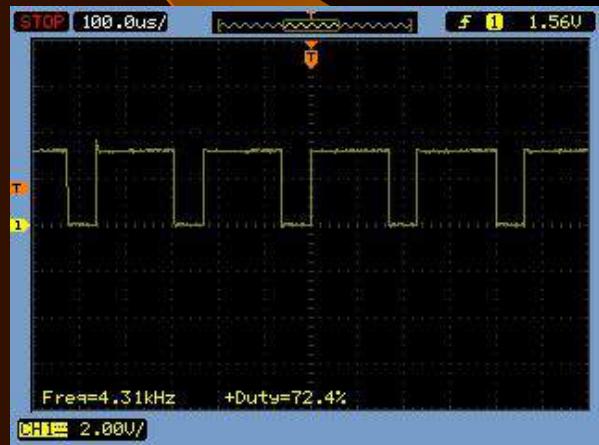
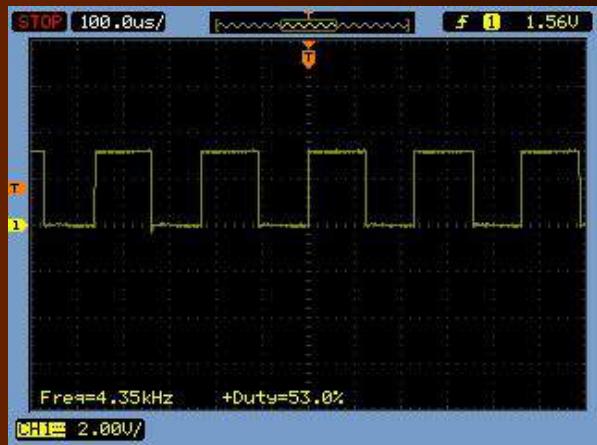
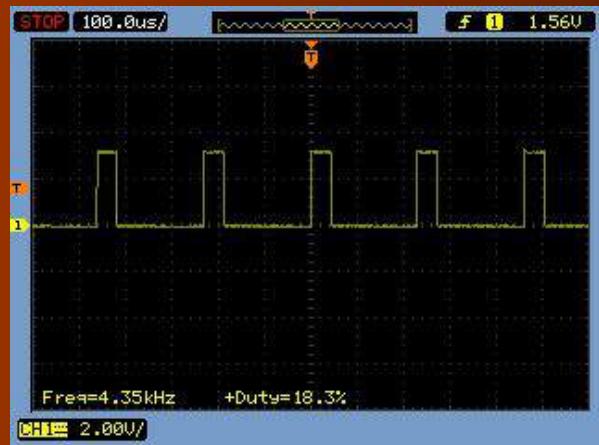


$R = 470 \text{ ohm}$, $C = 2.2 \mu\text{F}$

Filter Frequency = $1/(2 \cdot 3.14 \cdot R \cdot C) = 154 \text{ Hz}$

PWM signal at pin 1.6

Duty cycle of 18.3 %, 53 %, 72.4 %



Hardware PWM Code 2

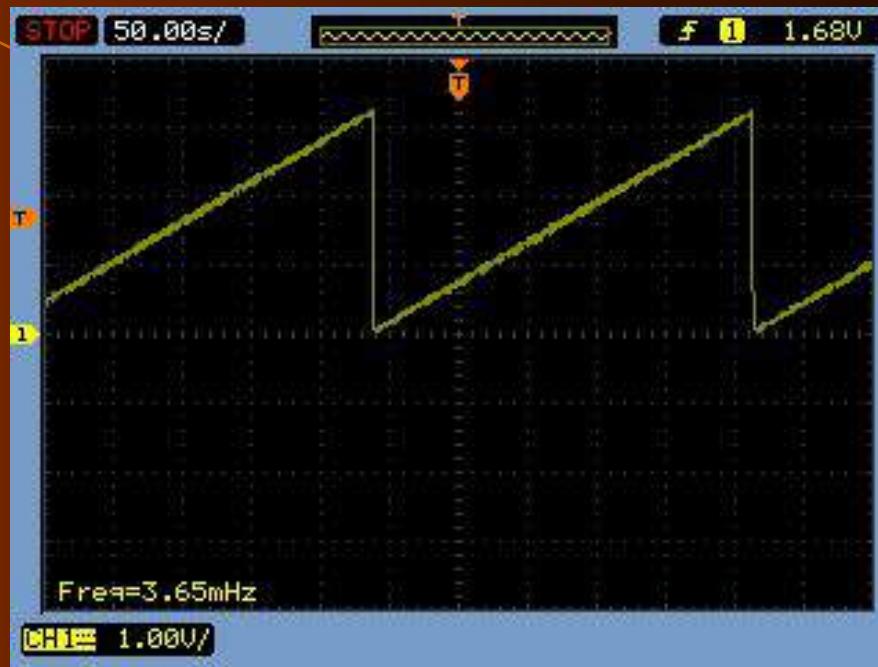
16 bit PWM at DCO freq

- 16 MHz

```
1 #include <msp430.h>
2
3 #define GREEN BIT6           // Green LED -> P1.6
4
5 /**
6 * @brief
7 * These settings are w.r.t enabling TIMER0 on Lunchbox
8 */
9 void register_settings_for_TIMER0()
10 {
11     P1DIR |= GREEN;          // Green LED -> Output
12     P1SEL |= GREEN;          // Green LED -> Select Timer Output
13
14     CCR0 = 65535;            // Set Timer0 PWM Period
15     CCTL1 = OUTMOD_7;        // Set TA0.1 Waveform Mode - Clear on Compare, Set on Overflow
16     CCR1 = 0;                // Set TA0.1 PWM duty cycle
17     CCTL0 = CCIE;            // CCR0 Enable Interrupt
18     TACTL = TASSEL_2 + MC_1; // Timer Clock -> SMCLK, Mode -> Up Count
19 }
20
```

```
28
29 /**
30 * @brief
31 * Entry point for the code
32 */
33 void main(void) {
34
35     WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer
36     BCSCTL1 |= (BIT0 + BIT1 + BIT2 + BIT3); // Selecting RSELX as 15
37     DCOCTL  |= (BIT6 + BIT5 + BIT4);      // Selecting DCOX as 7, DCO_freq = 15.6 MHz, Room Temp. ~ 25 deg. Celsius, Operating voltage 3.3 V
38
39     register_settings_for_TIMER0();
40
41 /**
42 * @brief
43 * Entry point for TIMER0_INTERRUPT vector
44 */
45 #pragma vector = TIMER0_A0_VECTOR
46 __interrupt void Timer_A(void)
47 {
48     CCR1 = CCR1 + 1;                  // Increment CCR1
49 }
```

Filtered Output

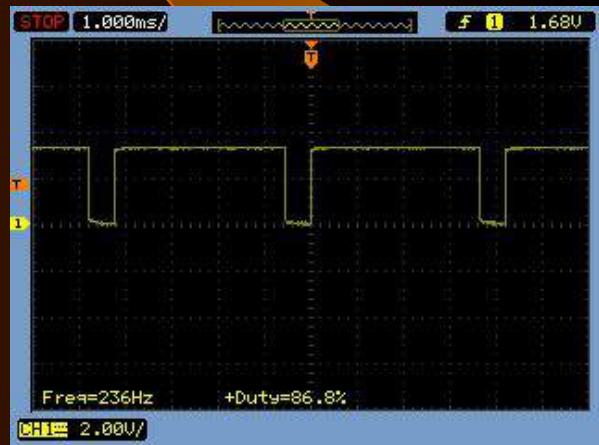
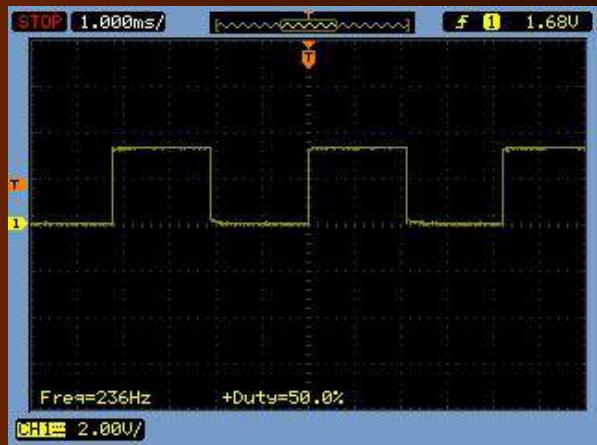
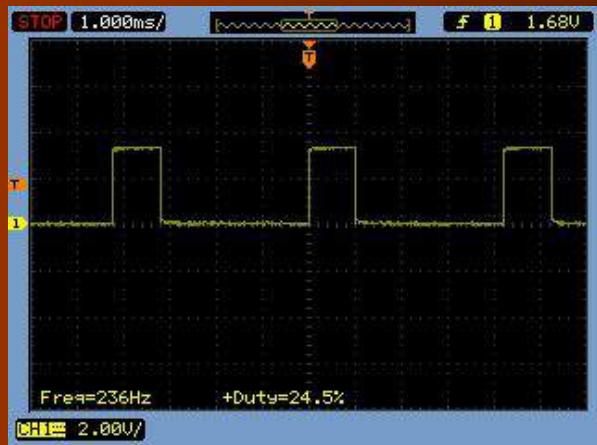


$R = 2.2 \text{ Kohm}$, $C = 22 \mu\text{F}$

Filter Frequency = $1/(2*3.14*R*C) = 3.28 \text{ Hz}$

PWM signal at pin 1.6

Duty cycle of 24.5 %, 50 %, 86.8 %





Thank you!

Introduction to Embedded System Design

Analog to Digital Converter

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

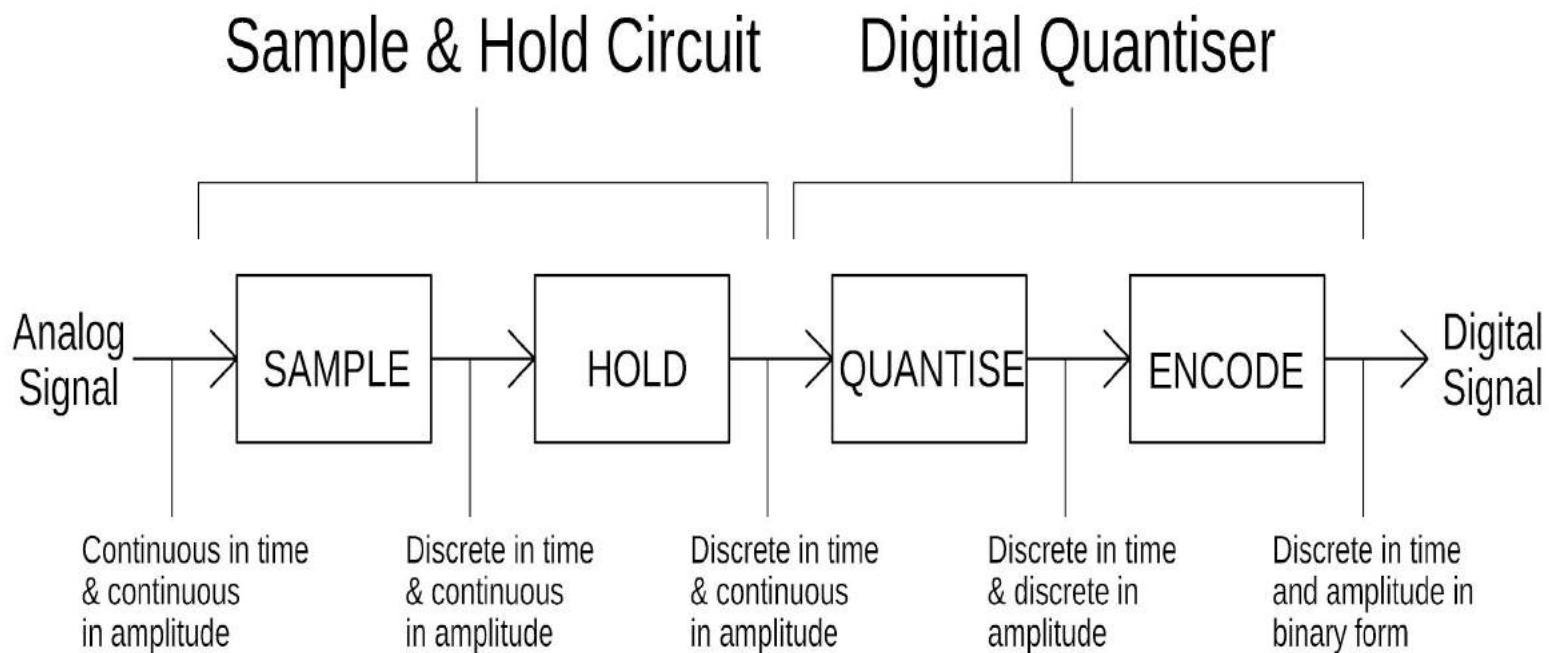
Electrical Engineering Department

Indian Institute of Technology,
Jammu

Why ADCs

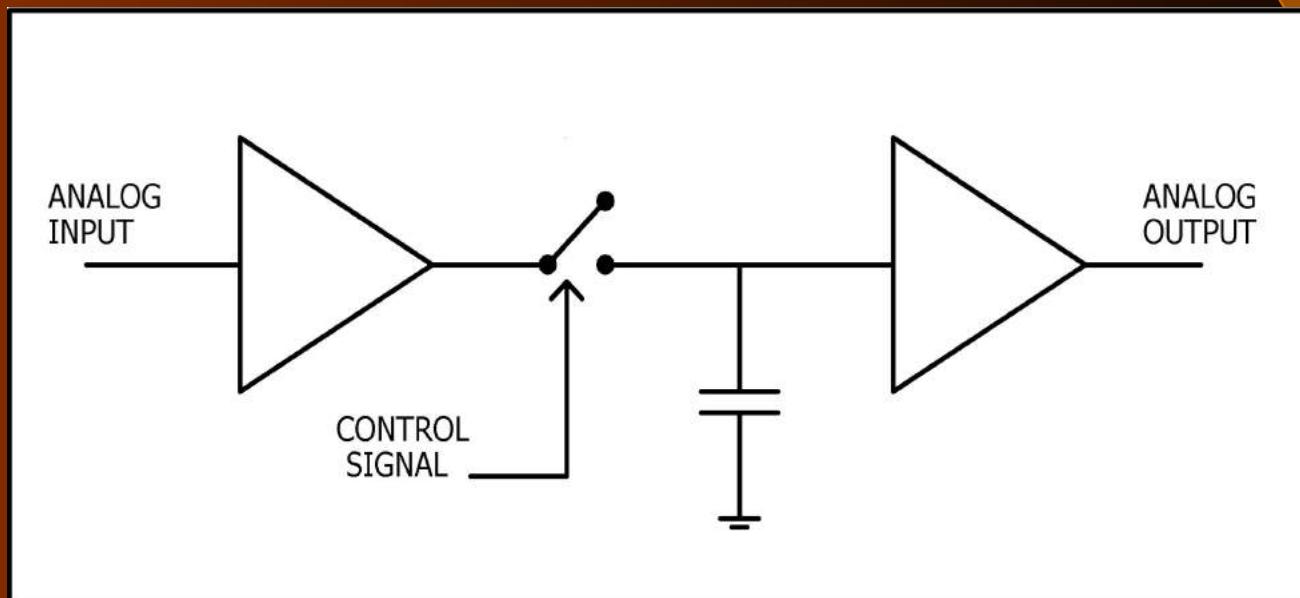
- The world is analog! All the real time signals like temperature, pressure, distance, or light intensity are continuous in nature. Most real world sensors provide analog data.
- Examples of sensors that provide analog values are: potentiometer, LDR (Light dependent resistor), NTC- thermistor, microphone, load cell etc.
- The CPU, however, can only process digital data and discrete values.
- It is required to map such analog inputs to some digital values. Hence, Analog to Digital Converters are used for this.

Basic Function of ADC



Sample And Hold Circuit

- Sample and hold circuit is required to sample analog signal (voltage) at a particular intervals depending upon timing of the control signal. The sampling rate has to be higher than Nyquist Rate. The sampled value is held for a specified minimum period of time. This is done to remove the variations in signal which can corrupt the conversion process.



Quantization

- After the sample and hold process, the signal becomes discrete in time but is still continuous in amplitude. The set of continuous values is mapped into a set of discrete values by the process of quantisation.
- The reference voltage range is divided into a set of ‘quantas’ and each sample is assigned a particular quanta.
- A small range of continuous values is mapped to a single value which introduces some error which is known as quantization error.

Encoding

- The process of assigning binary code to the quantised levels is known as encoding
- For encoding, every quantised level is given a unique digital value which is linear in nature with the quantised level
- The analog value is estimated with the help of quantization size and the digital value. The resolution of an ADC represents the number of distinct digital output values the ADC can give. For example, suppose the reference voltage is 3.3V, and the resolution is 10 bits. Then the number of outputs is $2^{10} = 1024$. Then, each sample from 0V to 3.3V can be mapped from 0 to 1023 linearly. Therefore, the resolution is $3.3V/1024 = 3.222mV$.

Types of ADCs

- Counter-Type ADC
- Successive Approximation Register Type ADC
- Dual Slope ADC
- Flash Type ADC
- Sigma-Delta Type ADC

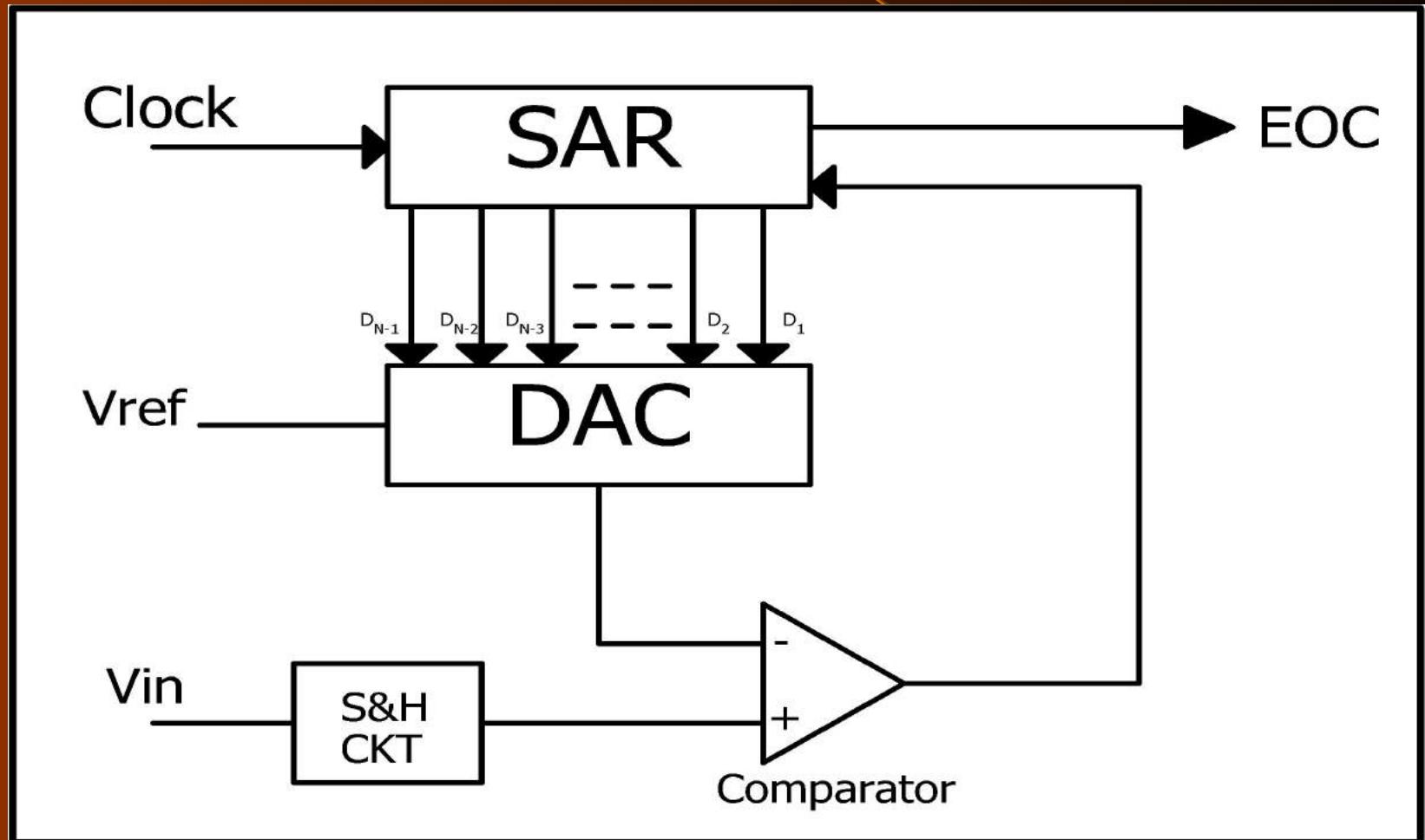
SAR - Successive Approximation Register

- This type of ADC uses a binary search approach for converting analog values into discrete values.
- It also has an internal comparator which compares the input voltage with the output of the digital - analog converter (DAC).
- Initially the MSB of the SAR is set and other bits are cleared. The SAR value is fed to the DAC and converted to the analog value. The input signal is now compared with the DAC value. If DAC value is less than the input voltage, the MSB is set low otherwise MSB is not changed.

SAR - Successive Approximation Register

- Now the next bit is set in the SAR. Again, the corresponding DAC analog output is compared with the input.
- Using the same procedure the next bits are calculated.
- After all the bits in SAR are tested, EOC(End Of Conversion) is reached and digital approximation of the analog input is given.

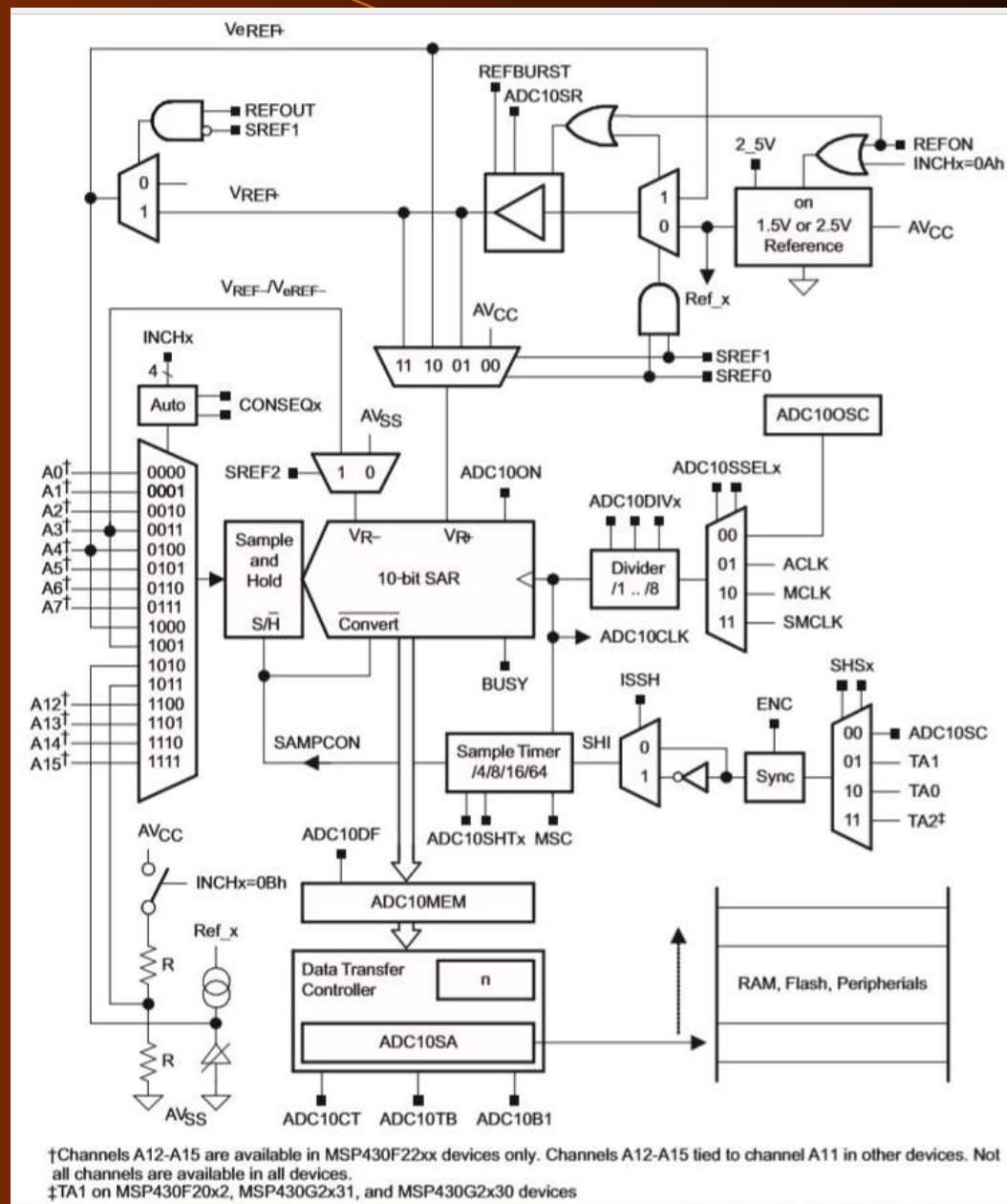
SAR - BLOCK DIAGRAM



Features of MSP430 ADC10

- The ADC provided in this controller is a 10-bit SAR type ADC.
- The module supports fast conversions with 200 ksps maximum conversions rate.
- The module implements a 10-bit SAR core, sample select control, reference generator, and data transfer controller (DTC).
- The DTC allows ADC10 samples to be converted and stored anywhere in memory without CPU intervention
- Sample-and-hold with programmable sample periods.
- Up to eight external input channels available
- Software selectable reference voltage , on-chip selection (1.5V or 2.5V) or external reference.
- ADC core and reference voltage can be powered down separately.

ADC10 Internal Block Diagram



ADC Operation

- The ADC core converts an analog input to its 10-bit digital representation and stores the result in the ADC10MEM register
- The reference voltages can be selected by the user
- The digital output is full scale when the input is higher or equal than the positive reference voltage and zero when the input signal is equal to or lower than negative reference voltage
- The conversion formula for the 10-bit ADC result when using straight binary format is:

$$N = 1023 * (V_{in} - V_{r-}) / (V_{r+} - V_{r-})$$

- The ADC10 source clock is selected using the ADC10SSELx bits and can be divided from 1 to 8 using the ADC10DIVx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK, and internal oscillator ADC10OSC .

Modes of operation

The ADC10 module has four operating modes which can be selected accordingly:

1. **Single channel single-conversion-** In this mode, only a single channel is converted once.
2. **Sequence-of-channels-** In this mode, a sequence of channels is converted once.
3. **Repeat single channel-** In this mode, a single channel is converted repeatedly.
4. **Repeat sequence-of-channels-** In this mode, a sequence of channels is converted repeatedly.

ADC10 Registers

- **ADC10CTL0** - Control register 0 of ADC10 selects the reference voltage, sample time, etc. It also enables the ADC Conversions and Interrupts.
- **ADC10CTL1** - Control register 1 of ADC10 selects the input channel, output data format, clock source and divider,
- **ADC10AE0** - Analog enable control register 0 enables the corresponding pin for analog input A0 - A7
- **ADC10AE1** - Analog enable control register 1 enables the corresponding pin for analog input A12-A15 (for the devices which have them)

ADC10 Registers Continued

- **ADC10MEM** - Conversion memory registers stores the output of the adc in either binary or 2's complement form which can be selected in ADC10CTL1
- **ADC10DTC0** - Data transfer control register 0 selects the mode of the DTC
- **ADC10DTC1** - Data transfer control register 1 selects the DTC transfers, i.e. it defines the number of transfers in each block
- **ADC10SA** - Start address register for data transfer selects the start address for the DTC. A write to register ADC10SA is required to initiate DTC transfers.

ADC10 Registers Memory Locations

Register	Short Form	Register Type	Address	Initial State
ADC10 input enable register 0	ADC10AE0	Read/write	04Ah	Reset with POR
ADC10 input enable register 1	ADC10AE1	Read/write	04Bh	Reset with POR
ADC10 control register 0	ADC10CTL0	Read/write	01B0h	Reset with POR
ADC10 control register 1	ADC10CTL1	Read/write	01B2h	Reset with POR
ADC10 memory	ADC10MEM	Read	01B4h	Unchanged
ADC10 data transfer control register 0	ADC10DTC0	Read/write	048h	Reset with POR
ADC10 data transfer control register 1	ADC10DTC1	Read/write	049h	Reset with POR
ADC10 data transfer start address	ADC10SA	Read/write	01BCh	0200h with POR

ADC10CTL0, ADC10 Control Register 0

ADC10CTL0 Continued

SREFx	Bits 15-13	Select reference.
	000	$V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$
	001	$V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$
	010	$V_{R+} = V_{eREF+}$ and $V_{R-} = V_{SS}$. Devices with V_{eREF+} only.
	011	$V_{R+} = \text{Buffered } V_{eREF+}$ and $V_{R-} = V_{SS}$. Devices with V_{eREF+} pin only.
	100	$V_{R+} = V_{CC}$ and $V_{R-} = V_{REF+}/V_{eREF-}$. Devices with V_{eREF-} pin only.
	101	$V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF+}/V_{eREF-}$. Devices with $V_{eREF+/‐}$ pins only.
	110	$V_{R+} = V_{eREF+}$ and $V_{R-} = V_{REF+}/V_{eREF-}$. Devices with $V_{eREF+/‐}$ pins only.
	111	$V_{R+} = \text{Buffered } V_{eREF+}$ and $V_{R-} = V_{REF+}/V_{eREF-}$. Devices with $V_{eREF+/‐}$ pins only.
ADC10SHTx	Bits 12-11	ADC10 sample-and-hold time
	00	4 × ADC10CLKs
	01	8 × ADC10CLKs
	10	16 × ADC10CLKs
	11	64 × ADC10CLKs
ADC10SR	Bit 10	ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
	0	Reference buffer supports up to ~200 kspS
	1	Reference buffer supports up to ~50 kspS
REFOUT	Bit 9	Reference output
	0	Reference output off
	1	Reference output on. Devices with V_{eREF+}/V_{REF+} pin only.
REFBURST	Bit 8	Reference burst.
	0	Reference buffer on continuously
	1	Reference buffer on only during sample-and-conversion
MSC	Bit 7	Multiple sample and conversion. Valid only for sequence or repeated modes.
	0	The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.
	1	The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed

ADC10CTL0 Continued

REF2_5V	Bit 6	Reference-generator voltage. REFON must also be set.
	0	1.5 V
	1	2.5 V
REFON	Bit 5	Reference generator on
	0	Reference off
	1	Reference on
ADC10ON	Bit 4	ADC10 on
	0	ADC10 off
	1	ADC10 on
ADC10IE	Bit 3	ADC10 interrupt enable
	0	Interrupt disabled
	1	Interrupt enabled
ADC10IFG	Bit 2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed.
	0	No interrupt pending
	1	Interrupt pending
ENC	Bit 1	Enable conversion
	0	ADC10 disabled
	1	ADC10 enabled
ADC10SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.
	0	No sample-and-conversion start
	1	Start sample-and-conversion

ADC10CTL1, ADC10 Control Register

ADC10CTL1 Continued

INCHx	Bits 15-12	Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific data sheet.
	0000	A0
	0001	A1
	0010	A2
	0011	A3
	0100	A4
	0101	A5
	0110	A6
	0111	A7
	1000	V_{eREF+}
	1001	V_{REF+}/V_{eREF-}
	1010	Temperature sensor
	1011	$(V_{CC} - V_{SS}) / 2$
	1100	$(V_{CC} - V_{SS}) / 2$, A12 on MSP430F22xx devices
	1101	$(V_{CC} - V_{SS}) / 2$, A13 on MSP430F22xx devices
	1110	$(V_{CC} - V_{SS}) / 2$, A14 on MSP430F22xx devices
	1111	$(V_{CC} - V_{SS}) / 2$, A15 on MSP430F22xx devices
SHSx	Bits 11-10	Sample-and-hold source select.
	00	ADC10SC bit
	01	Timer_A.OUT1 ⁽¹⁾
	10	Timer_A.OUT0 ⁽¹⁾
	11	Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices) ⁽¹⁾
ADC10DF	Bit 9	ADC10 data format
	0	Straight binary
	1	2s complement
ISSH	Bit 8	Invert signal sample-and-hold
	0	The sample-input signal is not inverted.
	1	The sample-input signal is inverted.

ADC10CTL1 Continued

ADC10DIVx	Bits 7-5	ADC10 clock divider
	000	/1
	001	/2
	010	/3
	011	/4
	100	/5
	101	/6
	110	/7
	111	/8
ADC10SSELx	Bits 4-3	ADC10 clock source select
	00	ADC10OSC
	01	ACLK
	10	MCLK
	11	SMCLK
(1) Timer triggers are from Timer0_Ax if more than one timer module exists on the device.		
CONSEQx	Bits 2-1	Conversion sequence mode select
	00	Single-channel-single-conversion
	01	Sequence-of-channels
	10	Repeat-single-channel
	11	Repeat-sequence-of-channels
ADC10BUSY	Bit 0	ADC10 busy. This bit indicates an active sample or conversion operation
	0	No operation is active.
	1	A sequence, sample, or conversion is active.

ADC10AE0, Analog (Input) Enable Control Register 0

ADC10AE0x							
7	6	5	4	3	2	1	0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
ADC10AE0x	Bits 7-0 ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT0 corresponds to A0, BIT1 corresponds to A1, etc. The analog enable bit of not implemented channels should not be programmed to 1.						
	0 Analog input disabled 1 Analog input enabled						

ADC10AE1, Analog (Input) Enable Control Register 1

	7	6	5	4	3	2	1	0
	ADC10AE1x							
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
ADC10AE1x	Bits 7-4 ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT4 corresponds to A12, BIT5 corresponds to A13, BIT6 corresponds to A14, and BIT7 corresponds to A15. The analog enable bit of not implemented channels should not be programmed to 1.						Reserved	
	0 Analog input disabled 1 Analog input enabled							
Reserved	Bits 3-0 Reserved							

ADC10MEM, Conversion Memory Register, Binary Format

15	14	13	12	11	10	9	8
0	0	0	0	0	0	Conversion Results	
r0	r0	r0	r0	r0	r0	r	r
7	6	5	4	3	2	1	0
Conversion Results							
r	r	r	r	r	r	r	r
Conversion Results	Bits 15-0	The 10-bit conversion results are right justified, straight-binary format. Bit 9 is the MSB. Bits 15-10 are always 0.					

ADC10MEM, Conversion Memory Register 2s Complement Format

15	14	13	12	11	10	9	8
Conversion Results							
r	r	r	r	r	r	r	r
7	6	5	4	3	2	1	0
Conversion Results							
r	r	r0	r0	r0	r0	r0	r0
Conversion Results	Bits 15-0	The 10-bit conversion results are left-justified, 2s complement format. Bit 15 is the MSB. Bits 5-0 are always 0.					

ADC10DTC0, Data Transfer Control Register 0

	7	6	5	4	3	2	1	0
	Reserved				ADC10TB	ADC10CT	ADC10B1	ADC10FETCH
	r0	r0	r0	r0	rw-(0)	rw-(0)	r-(0)	rw-(0)
Reserved	Bits 7-4	Reserved. Always read as 0.						
ADC10TB	Bit 3	ADC10 two-block mode						
		0	One-block transfer mode					
		1	Two-block transfer mode					
ADC10CT	Bit 2	ADC10 continuous transfer						
		0	Data transfer stops when one block (one-block mode) or two blocks (two-block mode) have completed.					
		1	Data is transferred continuously. DTC operation is stopped only if ADC10CT cleared, or ADC10SA is written to.					
ADC10B1	Bit 1	ADC10 block one. This bit indicates for two-block mode which block is filled with ADC10 conversion results. ADC10B1 is valid only after ADC10IFG has been set the first time during DTC operation. ADC10TB must also be set.						
		0	Block 2 is filled					
		1	Block 1 is filled					
ADC10FETCH	Bit 0	This bit should normally be reset.						

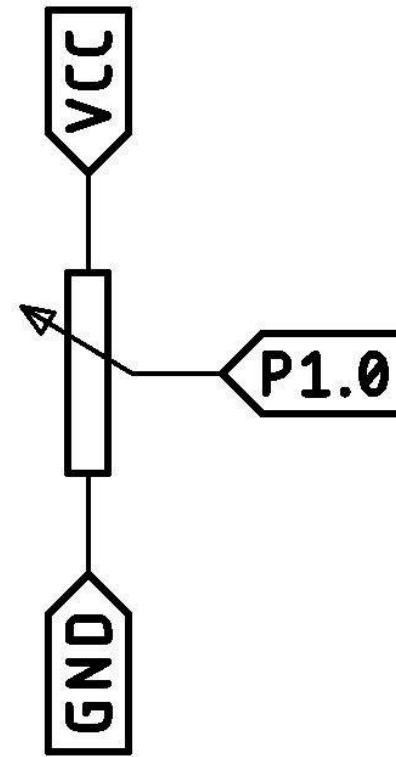
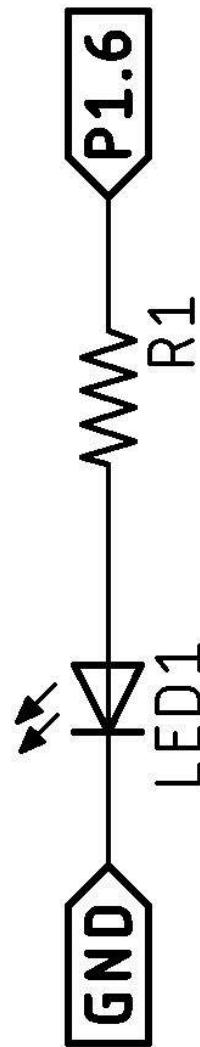
ADC10DTC1, Data Transfer Control Register 1

7	6	5	4	3	2	1	0
DTC Transfers							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
DTC Transfers		Bits 7-0 DTC transfers. These bits define the number of transfers in each block.					
		0 DTC is disabled					
		01h-0FFh Number of transfers per block					

ADC10SA, Start Address Register for Data Transfer

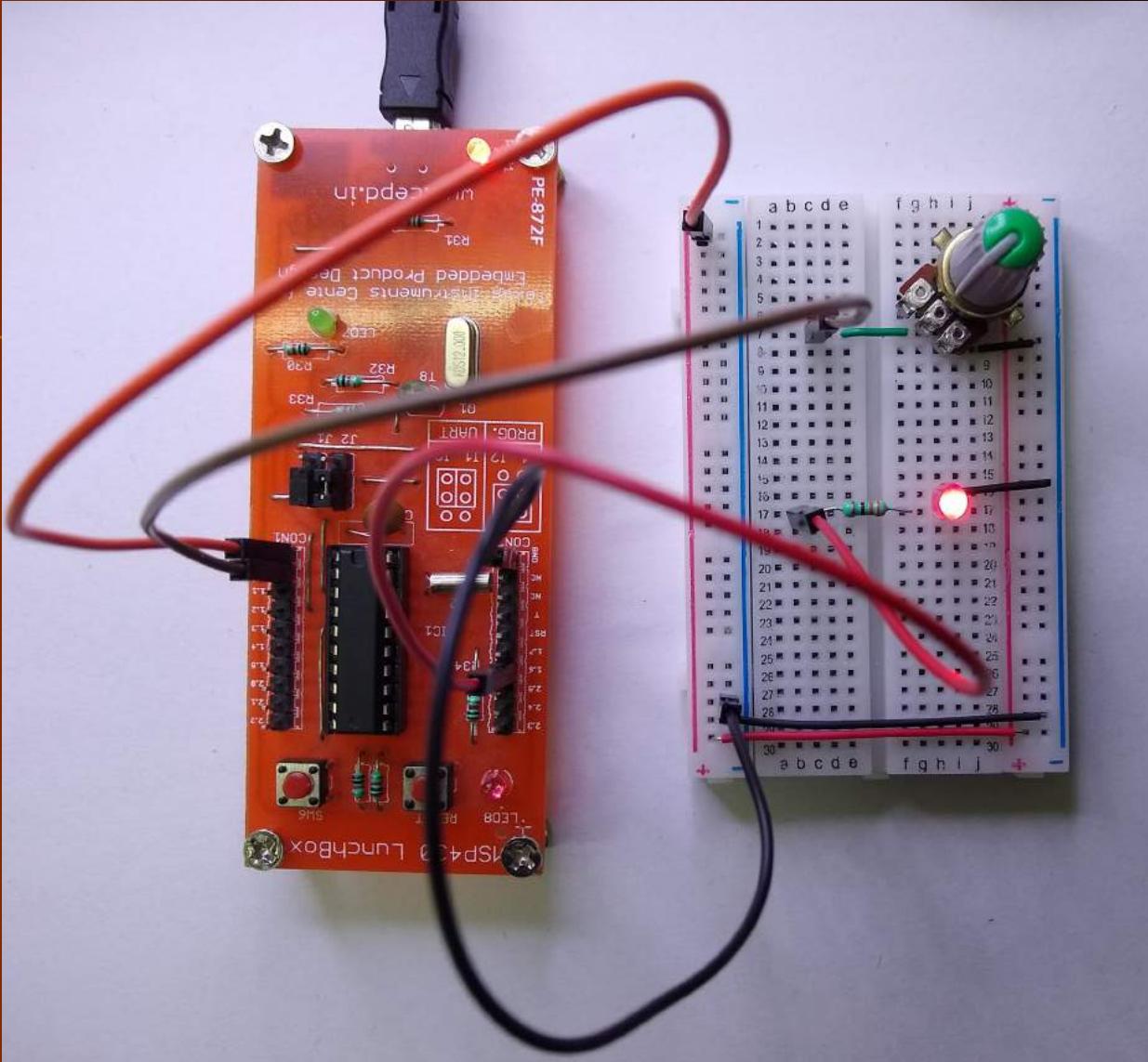
15	14	13	12	11	10	9	8
ADC10SAx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10SAx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0
ADC10SAx	Bits 15-1	ADC10 start address. These bits are the start address for the DTC. A write to register ADC10SA is required to initiate DTC transfers.					
Unused	Bit 0	Unused, Read only. Always read as 0.					

Code 1: Reading Analog value



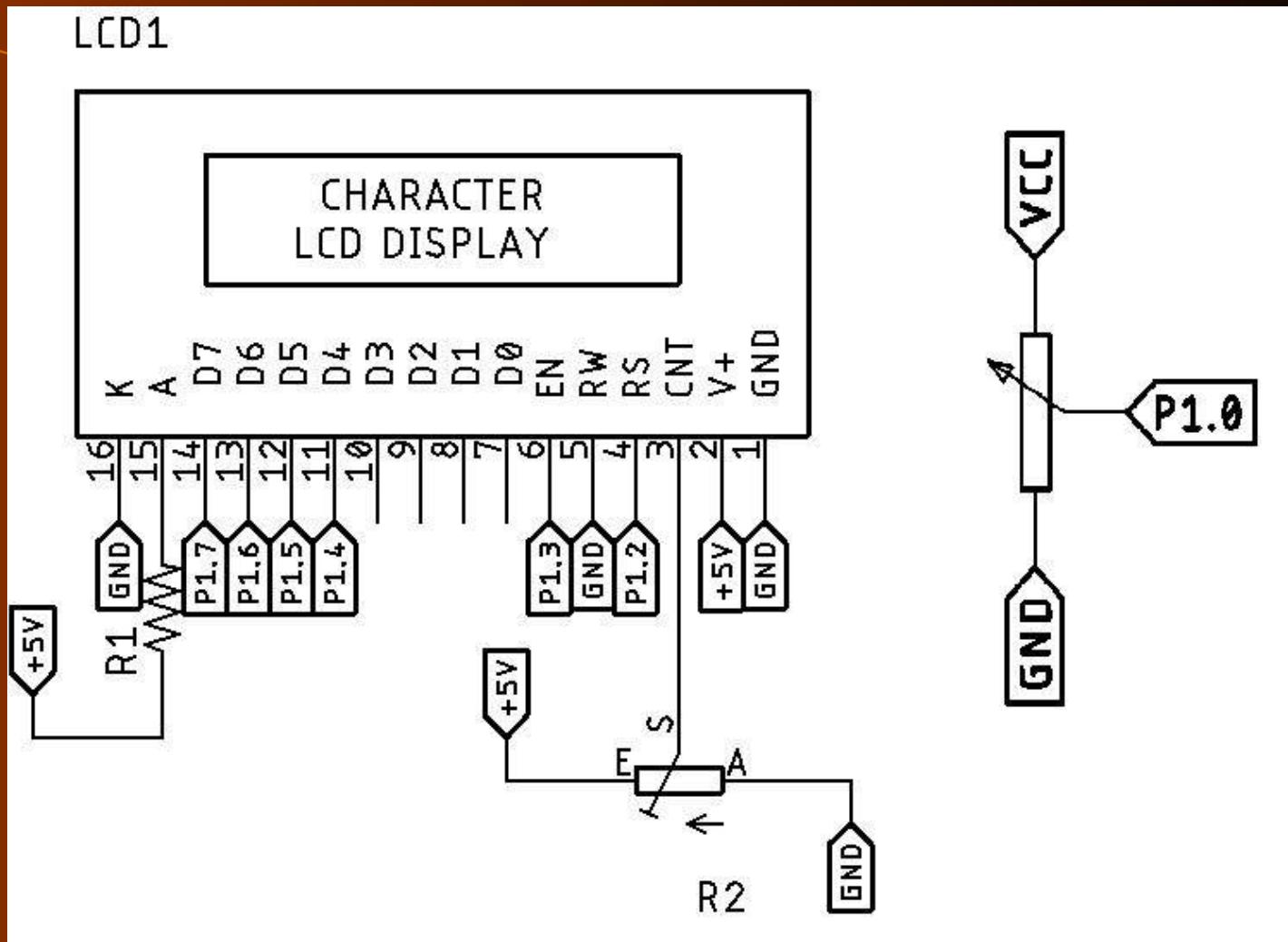
```
1 #include <msp430.h>
2
3 #define GREEN    BIT6
4 #define AIN      BIT0
5
6 /**
7  * @brief This function maps input range to the required range
8  * @param long Input value to be mapped
9  * @param long Input value range, minimum value
10 * @param long Input value range, maximum value
11 * @param long Output value range, minimum value
12 * @param long Output value range, maximum value
13 * @return Mapped output value
14 */
15 long map(long x, long in_min, long in_max, long out_min, long out_max) {
16     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
17 }
18
19 /**
20 * @brief
21 * These settings are wrt enabling ADC10 on Lunchbox
22 */
23 void register_settings_for_ADC10()
24 {
25     ADC10AE0 |= AIN;                                // P1.0 ADC option select
26     ADC10CTL1 = INCH_0;                            // ADC Channel -> 1 (P1.0)
27     ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON; // Ref -> Vcc, 64 CLK S&H
28 }
29
30 /**
31 * @brief
32 * These settings are wrt enabling TIMER0 on Lunchbox
33 */
34 void register_settings_for_TIMER0()
35 {
36     P1DIR |= GREEN;                                // Green LED -> Output
37     P1SEL |= GREEN;                                // Green LED -> Select Timer Output
38
39     CCR0 = 255;                                    // Set Timer0 PWM Period
40     CCTL1 = OUTMOD_7;                            // Set TA0.1 Waveform Mode
41     CCR1 = 1;                                     // Set TA0.1 PWM duty cycle
42     TACTL = TASSEL_2 + MC_1;                      // Timer Clock -> SMCLK, Mode -> Up Count
43 }
```

```
45 /*@brief entry point for the code*/
46 void main(void)
47 {
48     WDTCTL = WDTPW + WDTHOLD;           //!! Stop Watchdog (Not recommended for code in production and devices working in field)
49
50     register_settings_for_ADC10();
51
52     register_settings_for_TIMER0();
53
54
55     while(1)
56     {
57         ADC10CTL0 |= ENC + ADC10SC;       // Sampling and conversion start
58
59         while(ADC10CTL1 & ADC10BUSY);    // Wait for conversion to end
60
61         CCR1 = map(ADC10MEM, 0, 1023, 0, 255);
62     }
63 }
```



Potentiometer → P1.0
LED → P1.6

Code 2: Displaying Analog Value on the LCD

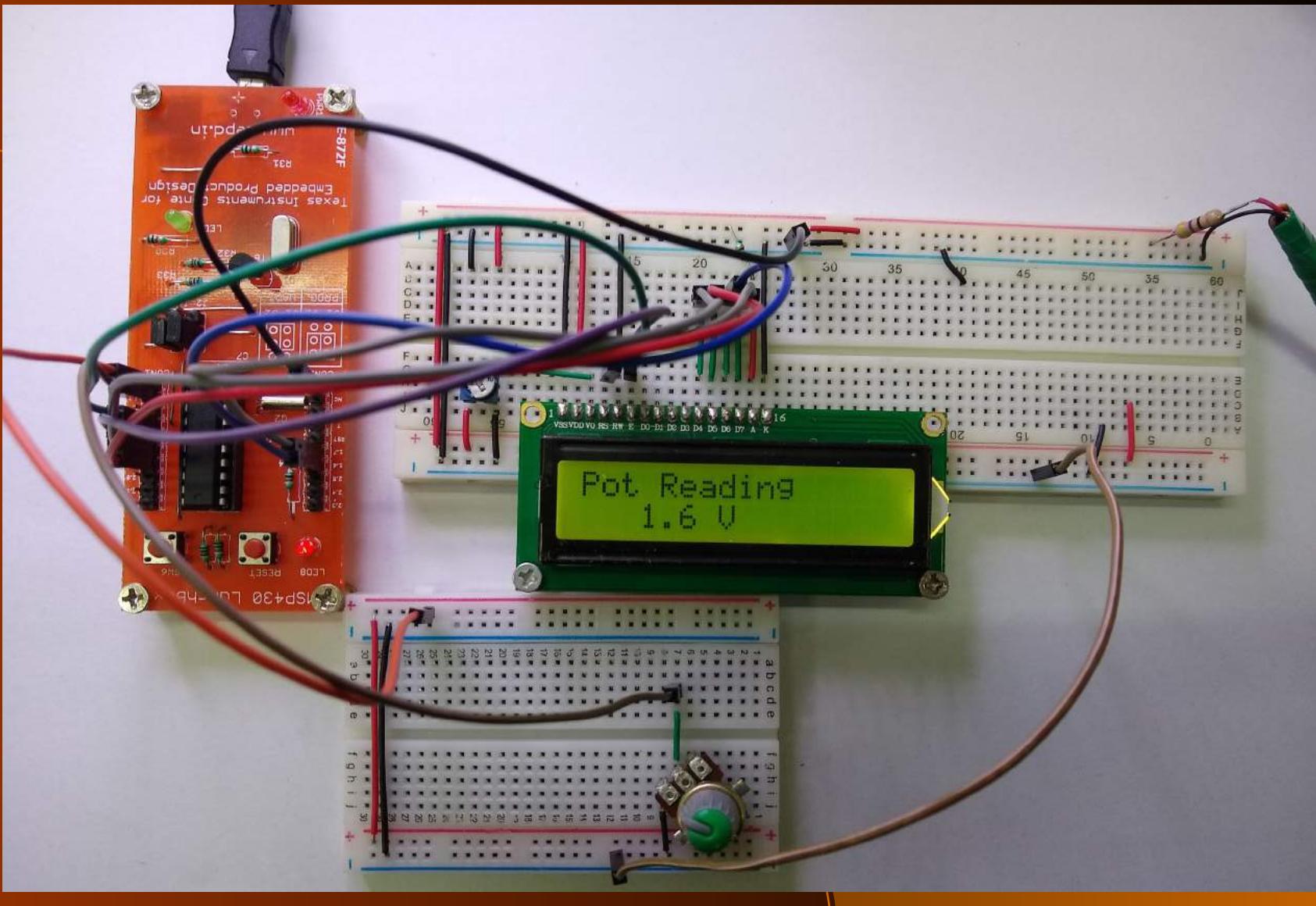


```
1 #include <msp430.h>
2 #include <inttypes.h>
3 #include<stdio.h>
4
5 #define CMD          0
6 #define DATA         1
7
8 #define AIN          BIT0           // Channel A0
9
10#define LCD_OUT       P1OUT
11#define LCD_DIR      P1DIR
12#define D4            BIT4
13#define D5            BIT5
14#define D6            BIT6
15#define D7            BIT7
16#define RS            BIT2
17#define EN            BIT3
18
19 /**
20 * @brief Delay function for producing delay in 0.1 ms increments
21 * @param t milliseconds to be delayed
22 * @return void
23 */
24 void delay(uint16_t t)
25 {
26     uint16_t i;
27     for(i=t; i > 0; i--)
28         __delay_cycles(100);
29 }
30
31 /**
32 * @brief Function to pulse EN pin after data is written
33 * @return void
34 */
35 void pulseEN(void)
36 {
37     LCD_OUT |= EN;
38     delay(1);
39     LCD_OUT &= ~EN;
40     delay(1);
41 }
```

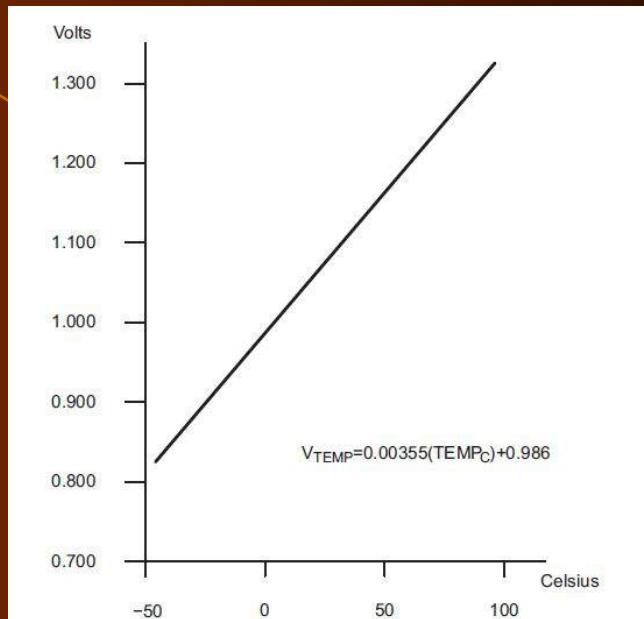
```
43 /**
44 * @brief Function to write data/command to LCD
45 * @param value Value to be written to LED
46 * @param mode Mode -> Command or Data
47 * @return void
48 */
49 void lcd_write(uint8_t value, uint8_t mode)
50 {
51     if(mode == CMD)
52         LCD_OUT &= ~RS;                      // Set RS -> LOW for Command mode
53     else
54         LCD_OUT |= RS;                     // Set RS -> HIGH for Data mode
55
56     LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));           // Write high nibble first
57     pulseEN();
58     delay(1);
59
60     LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));    // Write low nibble next
61     pulseEN();
62     delay(1);
63 }
64
65 /**
66 * @brief Function to print a string on LCD
67 * @param *s pointer to the character to be written.
68 * @return void
69 */
70 void lcd_print(char *s)
71 {
72     while(*s)
73     {
74         lcd_write(*s, DATA);
75         s++;
76     }
77 }
78
79 /**
80 * @brief Function to move cursor to desired position on LCD
81 * @param row Row Cursor of the LCD
82 * @param col Column Cursor of the LCD
83 * @return void
84 */
85 void lcd_setCursor(uint8_t row, uint8_t col)
86 {
87     const uint8_t row_offsets[] = { 0x00, 0x40};
88     lcd_write(0x80 | (col + row_offsets[row]), CMD);
89     delay(1);
90 }
```

```
92 /**
93 * @brief Function to change numeric value into it's corresponding char array
94 * @param num Number which has to be displayed
95 * @return void
96 */
97 void lcd_printNumber(unsigned int num)
98 {
99     char buf[3]; // Creating a array of size 3
100    char *str = &buf[2]; // Initializing pointer to end of the array
101
102    *str = '\0'; // storing null pointer at end of string
103
104    do
105    {
106        unsigned long m = num; // Storing number in variable m
107        num /= 10; // Dividing number by 10
108        char c = (m - 10 * num) + '0'; // Finding least place value and adding it to get character value of digit
109        *--str = c; // Decrementing pointer value and storing character at that character
110    } while(num);
111
112    lcd_print(str);
113 }
114
115 /**
116 * @brief Initialize LCD
117 */
118 void lcd_init()
119 {
120     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
121     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
122
123     delay(150); // Wait for power up ( 15ms )
124     lcd_write(0x33, CMD); // Initialization Sequence 1
125     delay(50); // Wait ( 4.1 ms )
126     lcd_write(0x32, CMD); // Initialization Sequence 2
127     delay(1); // Wait ( 100 us )
128
129     // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
130
131     lcd_write(0x28, CMD); // 4 bit mode, 2 line
132     delay(1);
133
134     lcd_write(0x0C, CMD); // Display ON, Cursor OFF, Blink OFF
135     delay(1);
136
137     lcd_write(0x01, CMD); // Clear screen
138     delay(20);
139
140     lcd_write(0x06, CMD); // Auto Increment Cursor
141     delay(1);
142
143     lcd_setCursor(0,0); // Goto Row 1 Column 1
144 }
```

```
146 /**
147 * @brief
148 * These settings are w/rt enabling ADC10 on lunchbox
149 */
150 void register_settings_for_ADC10()
151 {
152     ADC10AE0 |= AIN;                                // P1.0 ADC option select
153     ADC10CTL1 = INCH_0;                            // ADC Channel -> 1 (P1.0)
154     ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON;    // Ref -> Vcc, 64 CLK S&H , ADC - ON
155 }
156
157 /*@brief entry point for the code*/
158 void main(void)
159 {
160     WDTCTL = WDTPW + WDTHOLD;                      //! Stop Watchdog (Not recommended for code in production and devices working in field)
161
162     lcd_init();                                     // Initializing LCD
163
164     register_settings_for_ADC10();
165
166
167     while(1)
168     {
169         ADC10CTL0 |= ENC + ADC10SC;                // Sampling and conversion start
170
171         while(ADC10CTL1 & ADC10BUSY);             // Wait for conversion to end
172
173
174         float adc_value = 0;
175
176         adc_value = (ADC10MEM) * (3.30) / (1023.00); // mapping 10-bit conversion result of ADC to corresponding voltage
177
178         int int_part = adc_value;                  // Integer part of calculated ADC value
179         int decimal_part = (adc_value - (float)int_part) * 10.0; // Decimal part of calculated ADC value
180
181         lcd_write(0x01, CMD);                     // Clear screen
182         delay(20);
183
184         lcd_setCursor(0,0);
185         lcd_print("Pot Reading");
186
187         lcd_setCursor(1,3);
188         lcd_printNumber(int_part);                // Displaying integer part of ADC value
189         lcd_print(".");
190         lcd_printNumber(decimal_part);           // Displaying decimal part of ADC value
191         lcd_setCursor(1,7);
192         lcd_print("V");
193         delay(6000);
194     }
195 }
```



Example Code 3: Reading Internal Temperature sensor value



- $\text{Temp} = (\text{Vtemp} - 0.986) / 0.00355$
- $\text{Vtemp} = \text{ADC10MEM} * 1.5 / 1023$
- $\text{Temp} = (\text{ADC10MEM}-673)*423 / 1023$

```
1 #include <msp430.h>
2 #include <inttypes.h>
3 #include<stdio.h>
4
5 #define CMD          0
6 #define DATA         1
7
8 #define LCD_OUT      P1OUT
9 #define LCD_DIR     P1DIR
10 #define D4          BIT4
11 #define D5          BIT5
12 #define D6          BIT6
13 #define D7          BIT7
14 #define RS          BIT2
15 #define EN          BIT3
16
17 /**
18 * @brief Delay function for producing delay in 0.1 ms increments
19 * @param t milliseconds to be delayed
20 * @return void
21 */
22 void delay(uint16_t t)
23 {
24     uint16_t i;
25     for(i=t; i > 0; i--)
26         __delay_cycles(100);
27 }
28
29 /**
30 * @brief Function to pulse EN pin after data is written
31 * @return void
32 */
33 void pulseEN(void)
34 {
35     LCD_OUT |= EN;
36     delay(1);
37     LCD_OUT &= ~EN;
38     delay(1);
39 }
```

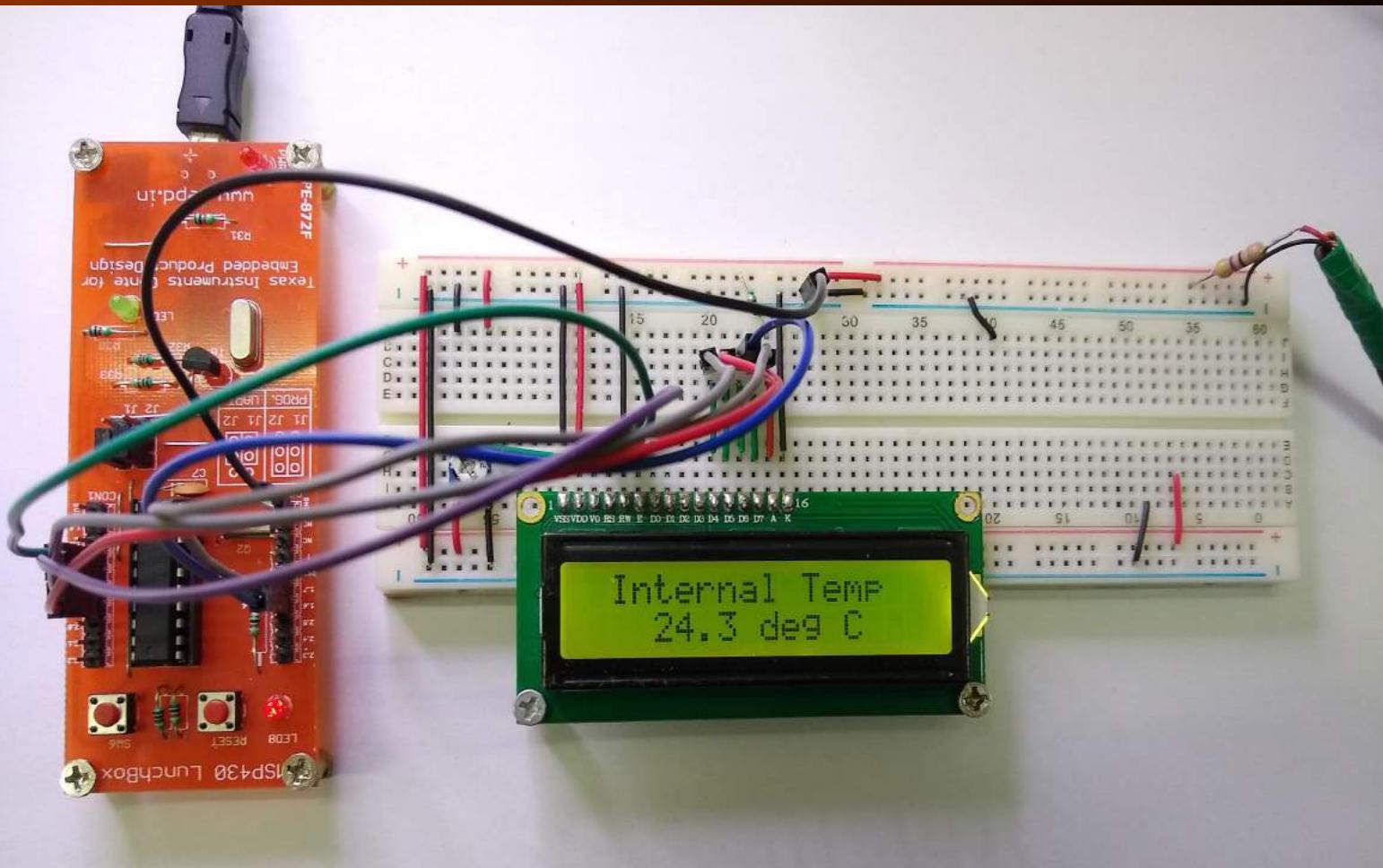
```
41 /**
42 * @brief Function to write data/command to LCD
43 * @param value Value to be written to LED
44 * @param mode Mode -> Command or Data
45 * @return void
46 */
47 void lcd_write(uint8_t value, uint8_t mode)
48 {
49     if(mode == CMD)
50         LCD_OUT &= ~RS;                      // Set RS -> LOW for Command mode
51     else
52         LCD_OUT |= RS;                     // Set RS -> HIGH for Data mode
53
54     LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));           // Write high nibble first
55     pulseEN();
56     delay(1);
57
58     LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));    // Write low nibble next
59     pulseEN();
60     delay(1);
61 }
62
63 /**
64 * @brief Function to print a string on LCD
65 * @param *s pointer to the character to be written.
66 * @return void
67 */
68 void lcd_print(char *s)
69 {
70     while(*s)
71     {
72         lcd_write(*s, DATA);
73         s++;
74     }
75 }
76
77 /**
78 * @brief Function to move cursor to desired position on LCD
79 * @param row Row Cursor of the LCD
80 * @param col Column Cursor of the LCD
81 * @return void
82 */
83 void lcd_setCursor(uint8_t row, uint8_t col)
84 {
85     const uint8_t row_offsets[] = { 0x00, 0x40};
86     lcd_write(0x80 | (col + row_offsets[row]), CMD);
87     delay(1);
88 }
89
```

```
90 /**
91 * @brief Function to change numeric value into it's corresponding char array
92 * @param num Number which has to be displayed
93 * @return void
94 */
95 void lcd_printNumber(unsigned int num)
96 {
97     char buf[3];                      // Creating a array of size 3
98     char *str = &buf[2];              // Initializing pointer to end of the array
99
100    *str = '\0';                     // storing null pointer at end of string
101
102    do
103    {
104        unsigned long m = num;       // Storing number in variable m
105        num /= 10;                 // Dividing number by 10
106        char c = (m - 10 * num) + '0'; // Finding least place value and adding it to get character value of digit
107        *--str = c;                // Decrementing pointer value and storing character at that character
108    } while(num);
109
110    lcd_print(str);
111 }
112
113 /**
114 * @brief Initialize LCD
115 */
116 void lcd_init()
117 {
118     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
119     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
120
121     delay(150);                   // Wait for power up ( 15ms )
122     lcd_write(0x33, CMD);         // Initialization Sequence 1
123     delay(50);                   // Wait ( 4.1 ms )
124     lcd_write(0x32, CMD);         // Initialization Sequence 2
125     delay(1);                    // Wait ( 100 us )
126
127     // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
128
129     lcd_write(0x28, CMD);         // 4 bit mode, 2 line
130     delay(1);
131
132     lcd_write(0x0C, CMD);         // Display ON, Cursor OFF, Blink OFF
133     delay(1);
134
135     lcd_write(0x01, CMD);         // Clear screen
136     delay(20);
137
138     lcd_write(0x06, CMD);         // Auto Increment Cursor
139     delay(1);
140
141     lcd_setCursor(0,0);           // Goto Row 1 Column 1
142 }
143
```

```

144 /**
145 * @brief This function maps input range to the required range
146 * @param long Input value to be mapped
147 * @param long Input value range, minimum value
148 * @param long Input value range, maximum value
149 * @param long Output value range, minimum value
150 * @param long Output value range, maximum value
151 * @return Mapped output value
152 */
153 void register_settings_for_ADC10()
154 {
155     ADC10CTL1 = INCH_10 + ADC10SSEL_3; // ADC Channel -> 10 (Internal Temperature Sensor), SMCLK
156     ADC10CTL0 = SREF_1 + ADC10SHT_3 + ADC10ON + REFON; // Ref -> Vref+ (1.5V) , 64 CLK S&H, ADC-ON, Reference generator-ON
157 }
158
159 /*@brief entry point for the code*/
160 void main(void)
161 {
162     WDTCTL = WDTPW + WDTHOLD; //! Stop Watchdog (Not recommended for code in production and devices working in field)
163
164     lcd_init();
165
166     register_settings_for_ADC10();
167
168
169     while(1)
170     {
171
172         ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
173         //lcd_display();
174         while(ADC10CTL1 & ADC10BUSY); // Wait for conversion to end
175
176         float temp = 0;
177         float Int_Deg_C = 0;
178
179         temp = ADC10MEM; // Storing 10-bit conversion result of ADC in variable temp
180
181         Int_Deg_C = ((temp-673)*423) / 1023; // Formula to find internal temperature in Celsius
182
183         int int_part = Int_Deg_C; // Integer part of calculated temperature value
184         int decimal_part = (Int_Deg_C - (float)int_part) * 10.0; // Decimal part of calculated temperature value
185
186         lcd_write(0x01, CMD); // Clear screen
187         delay(20);
188         lcd_setCursor(0,1);
189         lcd_print("Internal Temp");
190         lcd_setCursor(1,3);
191         lcd_printNumber(int_part); // Displaying integer part of temperature
192         lcd_print(".");
193         lcd_printNumber(decimal_part); // Displaying decimal part of temperature
194         lcd_print(" deg C");
195         delay(6000);
196
197     }
198 }
199

```





Thank you!

Introduction to Embedded System Design

ADC and DAC (LFSR and R2R Ladder)

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Random Numbers Generator

- A Random Number Generator generates a sequence of numbers or symbols in random order, that cannot be reasonably predicted better than by a random chance.
- Random number generators can be true hardware random-number generators (HRNG), which generate genuinely random numbers, or pseudo-random number generators (PRNG), which generate numbers that look random, but are actually deterministic, and can be reproduced if the state of the PRNG is known.

Practical Applications and Uses

- Computer Simulation
- Gambling
- Cryptography
- Security Applications

and many more...

Generating Random Numbers in Embedded Systems

1. Using noise as a source (Hardware RNG)
2. Avalanche noise in Zener diode (Hardware RNG)
3. By switch press event (Hardware RNG)
4. By using Linear Feedback Shift Register (LFSR)
(Software RNG)
5. Using rand() function in high level programming
language (Software RNG)

Linear Feedback Shift Register

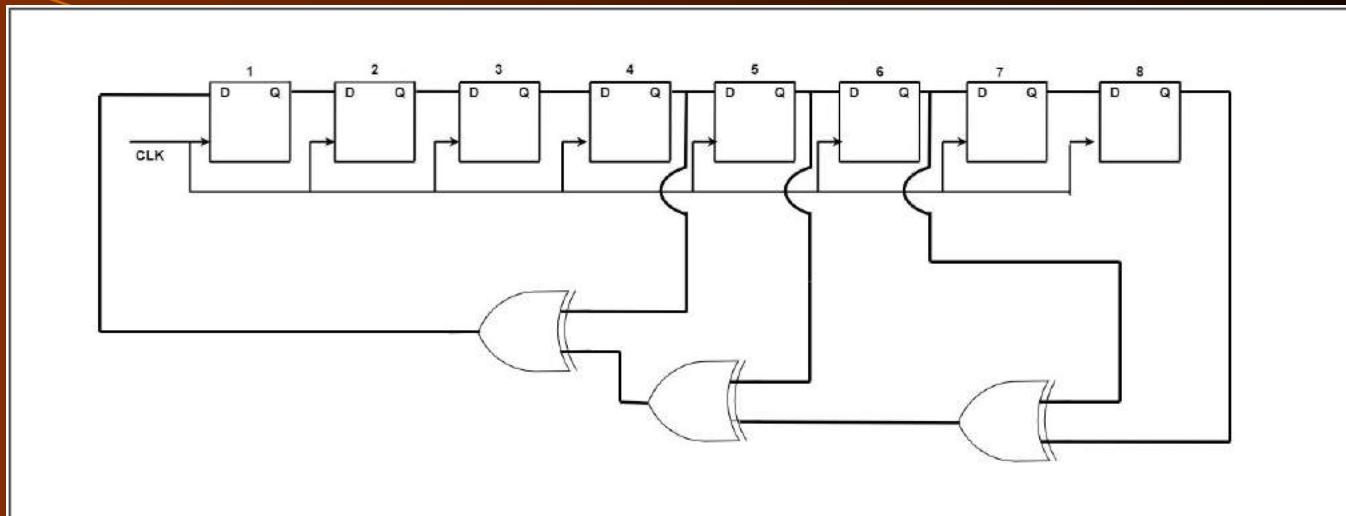
- A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.
- Most commonly used linear function of single bit is Exclusive-or (XOR).
- The initial value is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state.
- So, if one seed is used for two LFSRs, they produce the same sequence of numbers

Linear Feedback Shift Register

- Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle.
- However, an LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a very long cycle.
- The bits in the LFSR state that influence the input are called taps.

n	XNOR from	n	XNOR from	n	XNOR from	n	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

Linear Feedback Shift Register



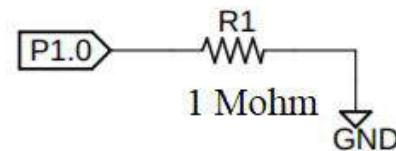
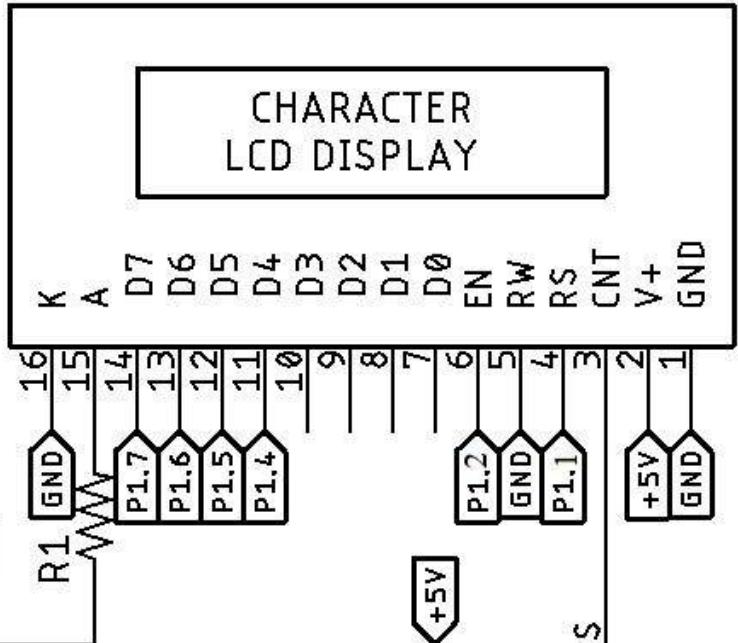
8 bit LFSR with taps at 8,6,5,4

Linear Feedback Shift Register

- An n bit LFSR can produce maximum $2^n - 1$ random numbers. (-1 for the combination with all zeros)
- Higher the value of n, more the number of random number outputs and more the time after which the random sequence repeats, more random the sequence appears.
- If the seed of the LFSR itself comes from a random source, like an analog input, the random numbers output becomes more random.
- Even an open Analog Pin gives random values due to noise.

LFSR Code Example 1

LCD1



On - board switch at pin P1.3

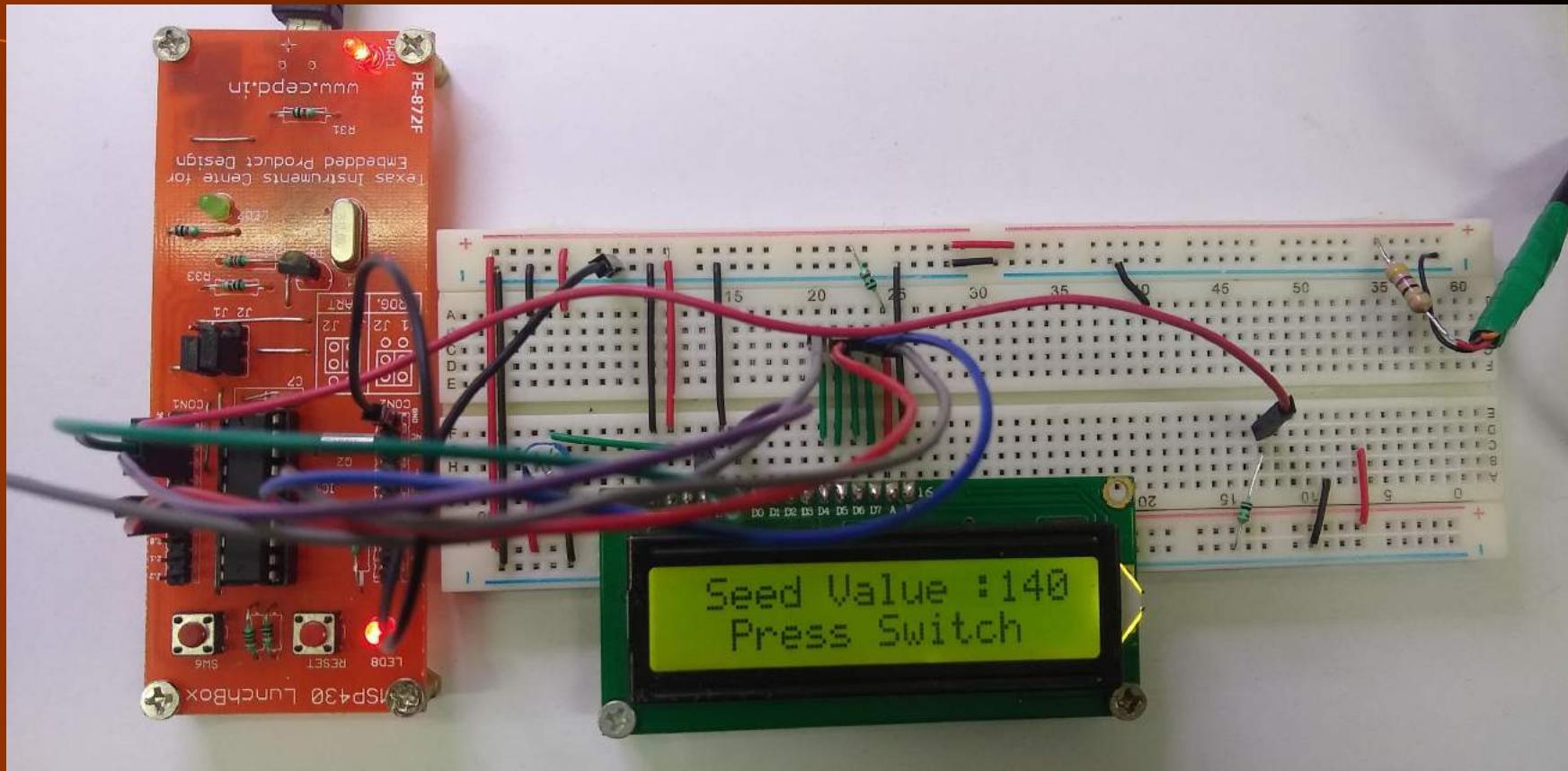
```
1 #include <msp430.h>
2 #include <inttypes.h>
3
4 #define SW  BIT3           // Switch -> P1.3 (On-board Switch, Pull-Up configuration)
5
6 #define CMD      0
7 #define DATA     1
8
9 #define AIN      BIT0        // Channel A0
10
11#define LCD_OUT   P1OUT
12#define LCD_DIR   P1DIR
13#define D4       BIT4
14#define D5       BIT5
15#define D6       BIT6
16#define D7       BIT7
17#define RS       BIT1
18#define EN       BIT2
19
20 /**
21 * @brief Delay function for producing delay in 0.1 ms increments
22 * @param t milliseconds to be delayed
23 * @return void
24 */
25 void delay(uint16_t t)
26 {
27     uint16_t i;
28     for(i=t; i > 0; i--)
29         __delay_cycles(100);
30 }
31
32 /**
33 * @brief Function to pulse EN pin after data is written
34 * @return void
35 */
36 void pulseEN(void)
37 {
38     LCD_OUT |= EN;
39     delay(1);
40     LCD_OUT &= ~EN;
41     delay(1);
42 }
```

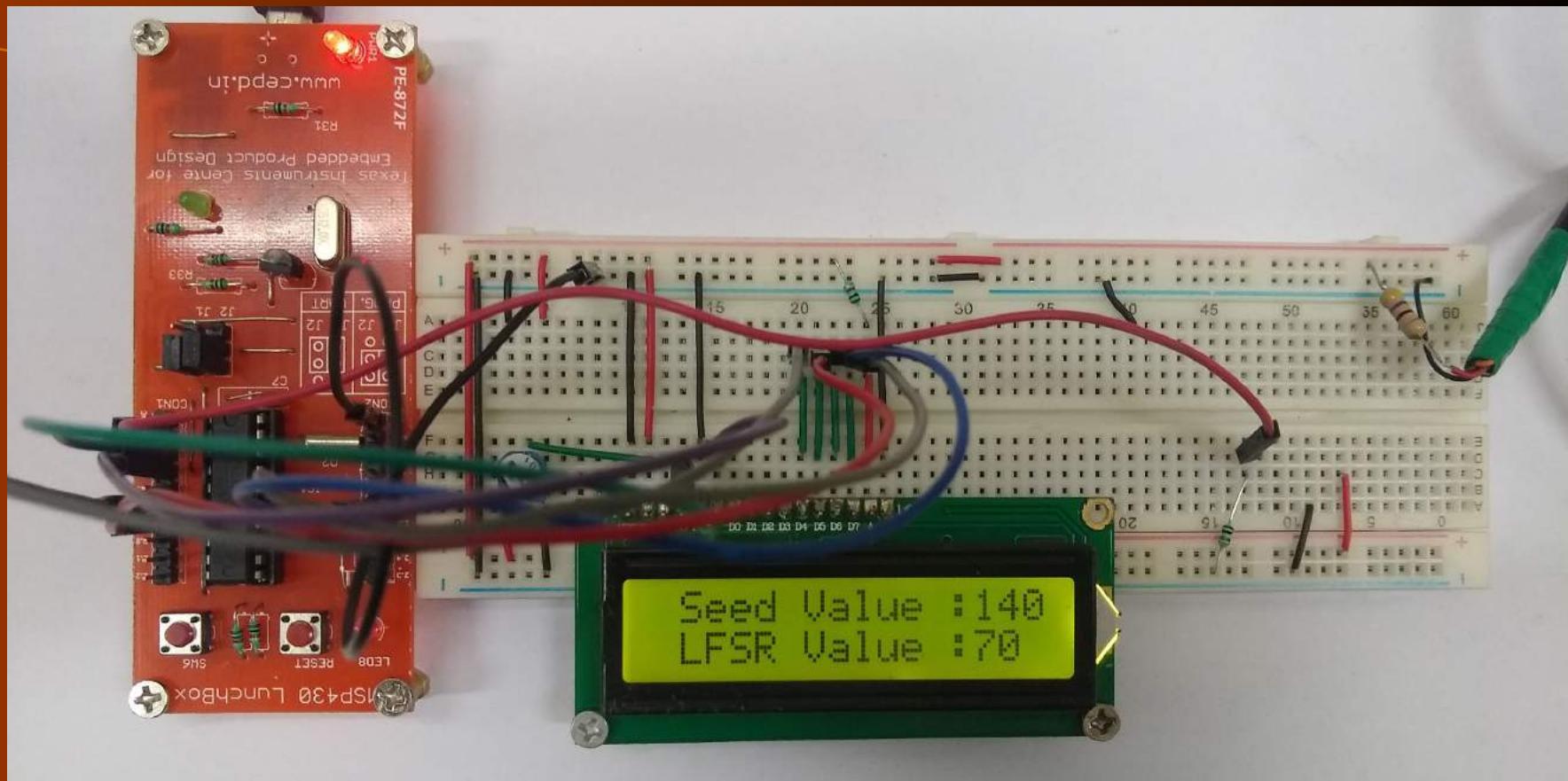
```
44 /**
45 * @brief Function to write data/command to LCD
46 * @param value Value to be written to LED
47 * @param mode Mode -> Command or Data
48 * @return void
49 */
50 void lcd_write(uint8_t value, uint8_t mode)
51{
52    if(mode == CMD)
53        LCD_OUT &= ~RS;                      // Set RS -> LOW for Command mode
54    else
55        LCD_OUT |= RS;                     // Set RS -> HIGH for Data mode
56
57    LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));           // Write high nibble first
58    pulseEN();
59    delay(1);
60
61    LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));      // Write low nibble next
62    pulseEN();
63    delay(1);
64}
65
66 /**
67 * @brief Function to print a string on LCD
68 * @param *s pointer to the character to be written.
69 * @return void
70 */
71 void lcd_print(char *s)
72{
73    while(*s)
74    {
75        lcd_write(*s, DATA);
76        s++;
77    }
78}
79
80 /**
81 * @brief Function to move cursor to desired position on LCD
82 * @param row Row Cursor of the LCD
83 * @param col Column Cursor of the LCD
84 * @return void
85 */
86 void lcd_setCursor(uint8_t row, uint8_t col)
87{
88    const uint8_t row_offsets[] = { 0x00, 0x40};
89    lcd_write(0x80 | (col + row_offsets[row]), CMD);
90    delay(1);
91}
92
```

```
93 /**
94 * @brief Function to change numeric value into it's corresponding char array
95 * @param num Number which has to be displayed
96 * @return void
97 */
98 void lcd_printNumber(unsigned int num)
99 {
100     char buf[4];           // Creating a array of size 4
101    char *str = &buf[3];   // Initializing pointer to end of the array
102
103    *str = '\0';           // storing null pointer at end of string
104
105    do
106    {
107        unsigned long m = num;          // Storing number in variable m
108        num /= 10;                    // Dividing number by 10
109        char c = (m - 10 * num) + '0'; // Finding least place value and adding it to get character value of digit
110        *--str = c;                  // Decrementing pointer value and storing character at that character
111    } while(num);
112
113    lcd_print(str);
114 }
115
116 /**
117 * @brief Initialize LCD
118 */
119 void lcd_init()
120 {
121     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
122     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
123
124     delay(150);                // Wait for power up ( 15ms )
125     lcd_write(0x33, CMD);      // Initialization Sequence 1
126     delay(50);                 // Wait ( 4.1 ms )
127     lcd_write(0x32, CMD);      // Initialization Sequence 2
128     delay(1);                  // Wait ( 100 us )
129
130     // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
131
132     lcd_write(0x28, CMD);      // 4 bit mode, 2 line
133     delay(1);
134
135     lcd_write(0x0C, CMD);      // Display ON, Cursor OFF, Blink OFF
136     delay(1);
137
138     lcd_write(0x01, CMD);      // Clear screen
139     delay(20);
140
141     lcd_write(0x06, CMD);      // Auto Increment Cursor
142     delay(1);
143
144     lcd_setCursor(0,0);        // Goto Row 1 Column 1
145 }
146
```

```
147 /**
148 * @brief
149 * These settings are w/ enabling ADC10 on Lunchbox
150 */
151 void register_settings_for_ADC10()
152 {
153     ADC10AE0 |= AIN;                                // P1.0 ADC option select
154     ADC10CTL1 = INCH_0;                            // ADC Channel -> 1 (P1.0)
155     ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON;    // Ref -> Vcc, 64 CLK S&H , ADC - ON
156 }
157
158 /*@brief entry point for the code*/
159 void main(void)
160 {
161     WDTCTL = WDTPW + WDTHOLD;                    // Stop Watchdog (Not recommended for code in production and devices working in field)
162
163     P1DIR &= ~SW;                                // Set SW pin -> Input
164
165     lcd_init();                                 // Initializing LCD
166
167     register_settings_for_ADC10();
168
169     while(1)
170     {
171         ADC10CTL0 |= ENC + ADC10SC;              // Sampling and conversion start
172
173         while(ADC10CTL1 & ADC10BUSY);           // Wait for conversion to end
174
175         uint8_t seed = (ADC10MEM & 0xFF);      // Lower 8-bits of ADC conversion result as seed
176         uint8_t lfsr = seed;
177         uint8_t count = 0;
178
179         lcd_write(0x01, CMD);                  // Clear screen
180         delay(20);
181
182         lcd_setCursor(0,1);
183         lcd_print("Seed Value :");
184         lcd_printNumber(seed);                // Printing Seed Value
185
186         lcd_setCursor(1,2);
187         lcd_print("Press Switch");
188 }
```

```
189     while(count<=255)          // Changing Seed Value after reading 255 LFSR values
190     {
191         if(!(P1IN & SW))      // If SW is Pressed
192         {
193             delay_cycles(20000); // Wait 20ms to debounce
194             while(!(P1IN & SW)); // Wait till SW Released
195             delay_cycles(20000); // Wait 20ms to debounce
196
197             uint8_t bit;
198             bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 4)); // Taking taps
199             lfsr = (lfsr >> 1) | (bit << 7); // Shifting LFSR
200
201             count = count + 1;
202
203             lcd_write(0x01, CMD); // Clear screen
204             delay(20);
205
206             lcd_setCursor(0,1);
207             lcd_print("Seed Value :");
208             lcd_printNumber(seed); // Printing Seed Value
209
210             lcd_setCursor(1,1);
211             lcd_print("LFSR Value :");
212             lcd_printNumber(lfsr); // Printing LFSR Value
213             delay(3000);
214         }
215     }
216
217 }
218 }
219 }
```





LFSR Code Example 2

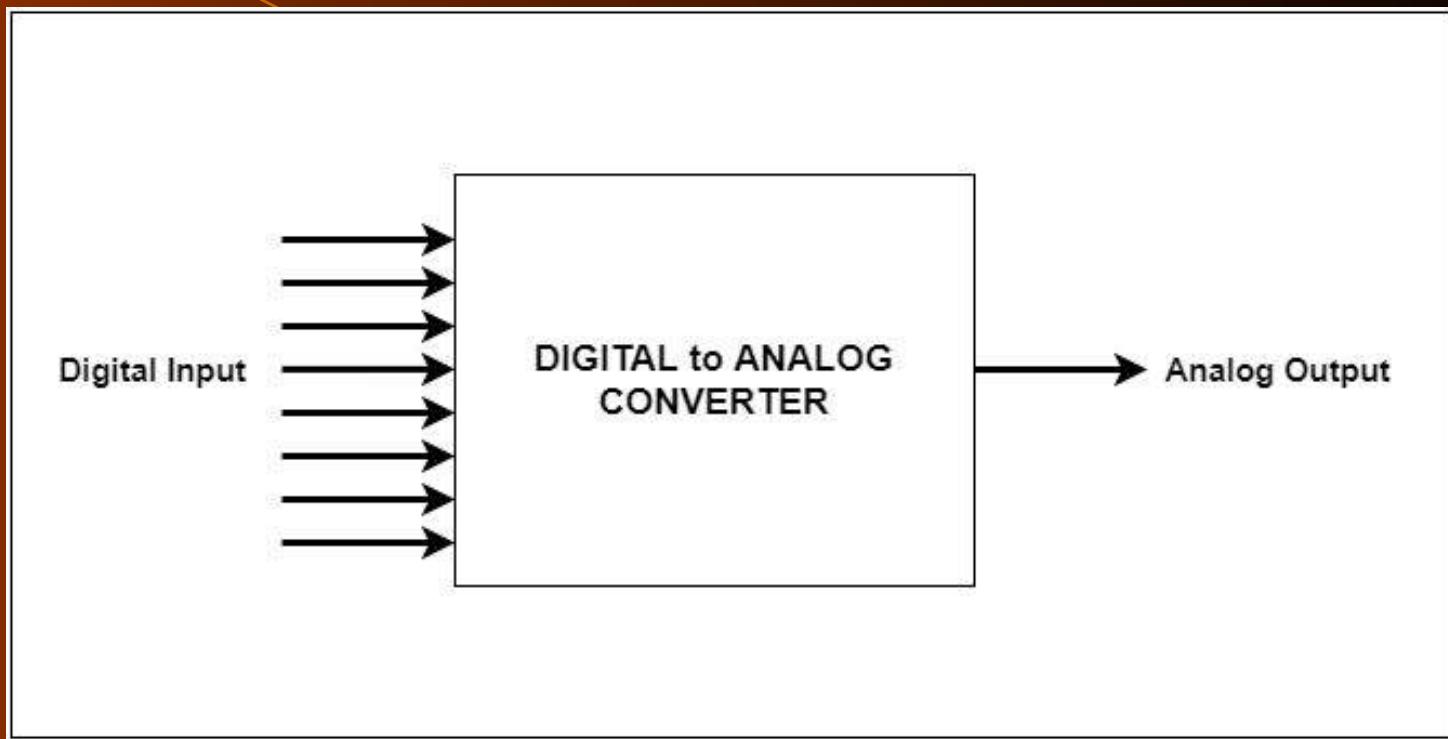
```
1 #include <msp430.h>
2 #include <inttypes.h>
3 #include<stdio.h>
4
5 #define RED  BIT7           // Switch -> P1.3 (On-board Switch, Pull-Up configuration)
6 #define AIN      BIT0       // Channel A0
7
8 /**
9 * @brief Delay function for producing delay in 0.1 ms increments
10 * @param t milliseconds to be delayed
11 * @return void
12 */
13 void delay(uint16_t t)
14 {
15     uint16_t i;
16     for(i=t; i > 0; i--)
17         __delay_cycles(100);
18 }
19
20
21 /**
22 * @brief
23 * These settings are wrt enabling ADC10 on Lunchbox
24 */
25 void register_settings_for_ADC10()
26 {
27     ADC10AE0 |= AIN;           // P1.0 ADC option select
28     ADC10CTL1 = INCH_0;        // ADC Channel -> 1 (P1.0)
29     ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON; // Ref -> Vcc, 64 CLK S&H , ADC - ON
30 }
31
```

```
32 /*@brief entry point for the code*/
33 void main(void)
34 {
35     WDTCTL = WDTPW + WDTHOLD;           //!! Stop Watchdog (Not recommended for code in production and devices working in field)
36
37     P1DIR |= RED;                     // P1.7 (Red LED)
38
39     register_settings_for_ADC10();
40
41     while(1)
42     {
43         ADC10CTL0 |= ENC + ADC10SC;      // Sampling and conversion start
44
45         while(ADC10CTL1 & ADC10BUSY);    // Wait for conversion to end
46
47         uint32_t lfsr = 0;
48         lfsr |= ADC10MEM;
49         uint32_t count = 0;
50
51         while(count < 0xFFFFFFFF)
52         {
53             uint32_t bit;
54             bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 6) ^ (lfsr >> 7));    // Taking taps
55             lfsr = (lfsr >> 1) | (bit << 31);          // Shifting LFSR
56
57             P1OUT = (lfsr & 0x00000001)<<7; ;        // Setting RED LED according to one bit of 32-bit LFSR
58             count = count + 1;
59             delay(150);
60         }
61     }
62 }
```

Digital to Analog Converter

- A digital-to-analog converter (DAC, D/A, D2A, or D-to-A) is a system that converts a digital signal into an analog signal. An analog-to-digital converter (ADC) performs the reverse function.

Block Diagram



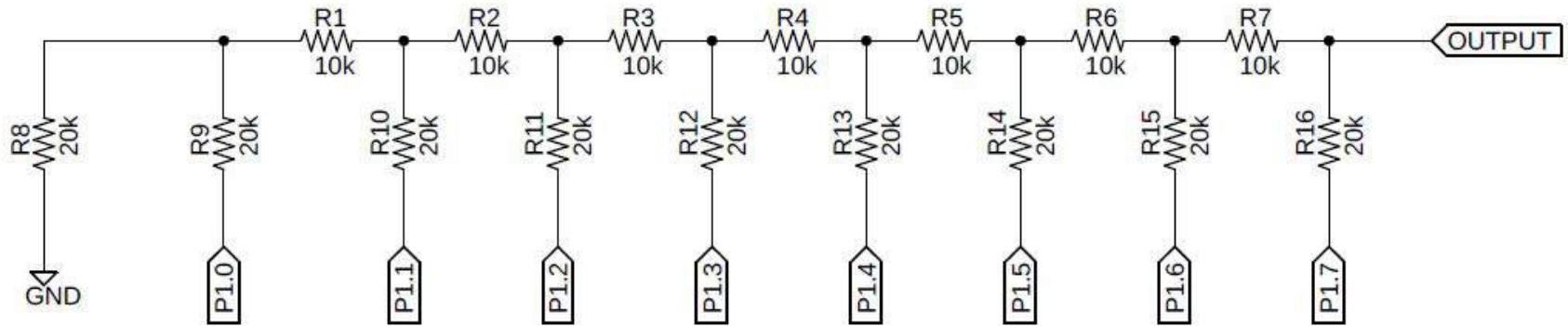
Types of DAC

- Pulse width modulation
- Binary weighted DAC
 - Switched Resistor
 - Switched Current Source
 - Switched Capacitor
 - R-2R ladder
- Successive approximation or cyclic DAC

R-2R Ladder DAC

- An R–2R ladder is a simple and inexpensive way to perform digital-to-analog conversion, using repetitive arrangements of precise resistor networks in a ladder-like configuration.

8 bit R-2R Ladder DAC

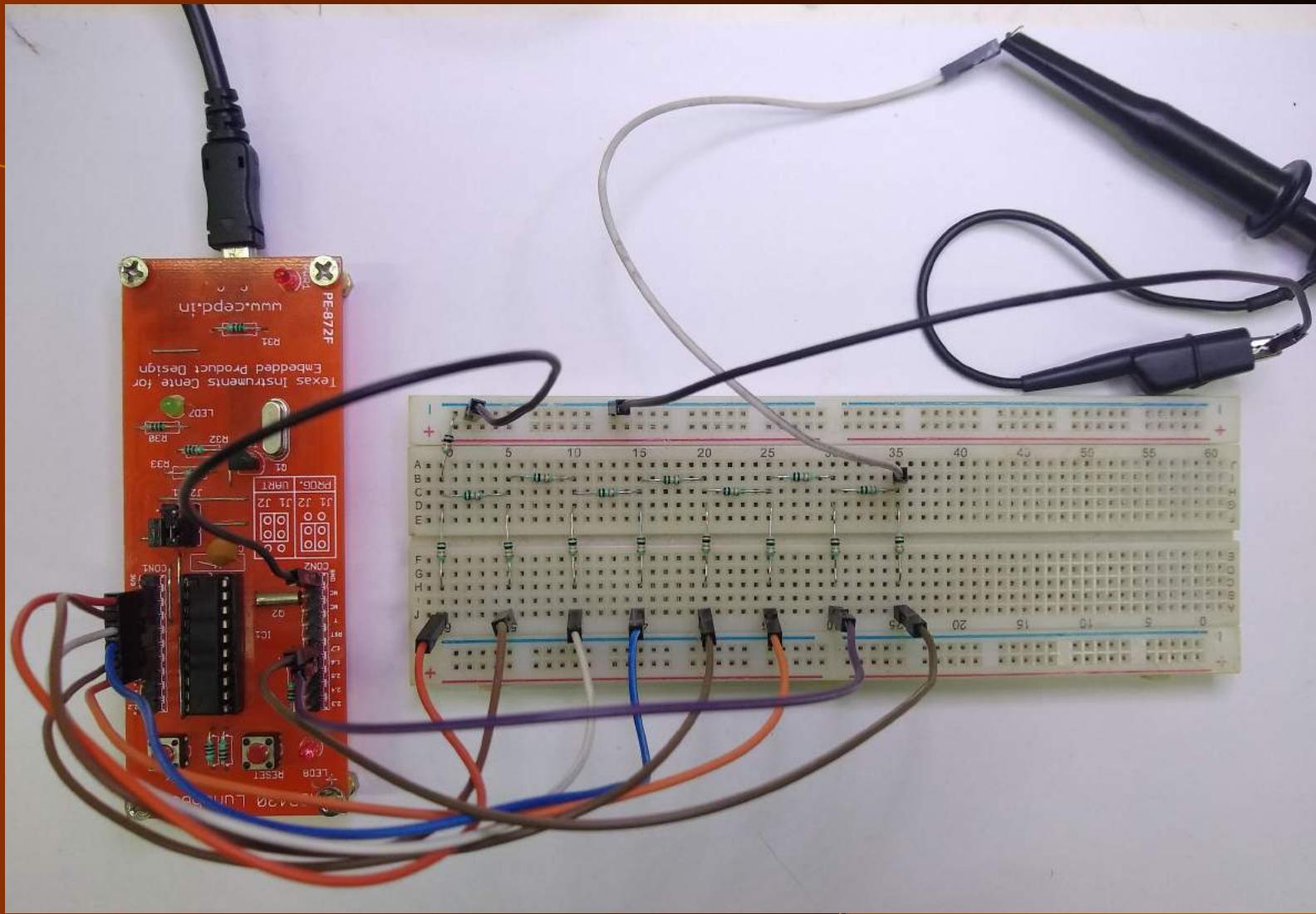


8 BIT DAC

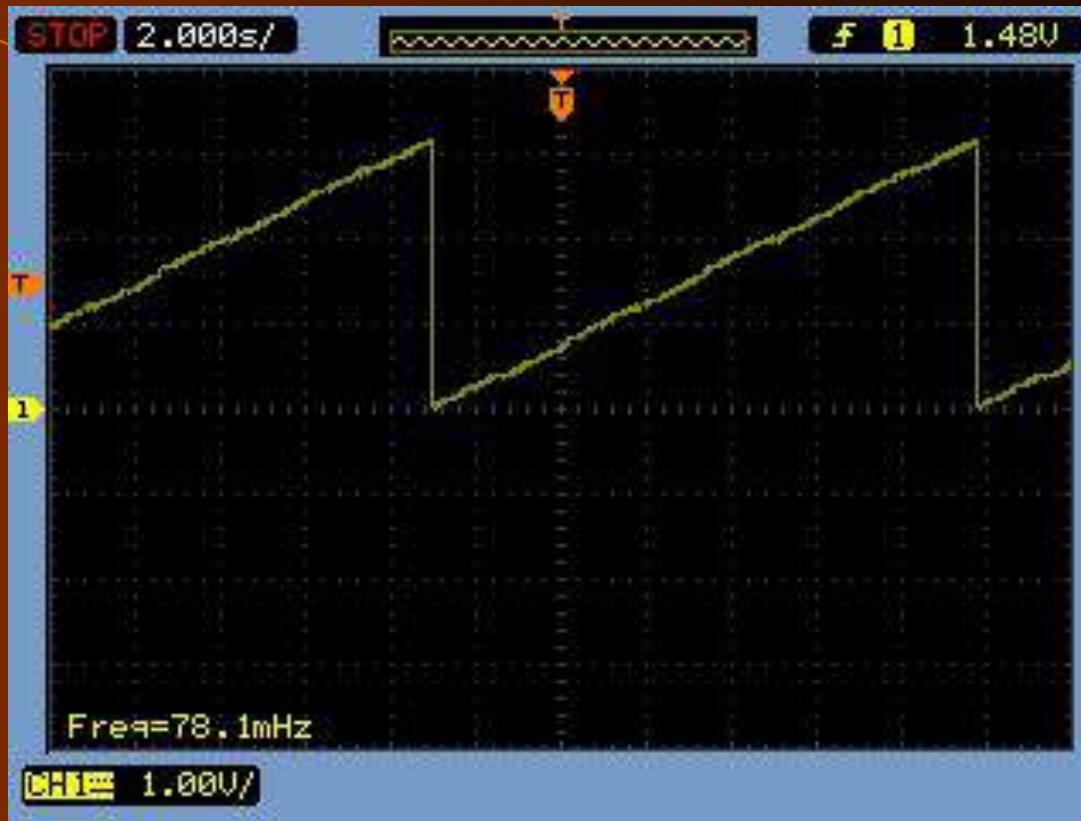
$$V_{out} = V_{b0} / 256 + V_{b1} / 128 + V_{b2} / 64 + V_{b3} / 32 + V_{b4} / 16 + V_{b5} / 8 + V_{b6} / 4 + V_{b7} / 2$$

Example Code : Hello DAC

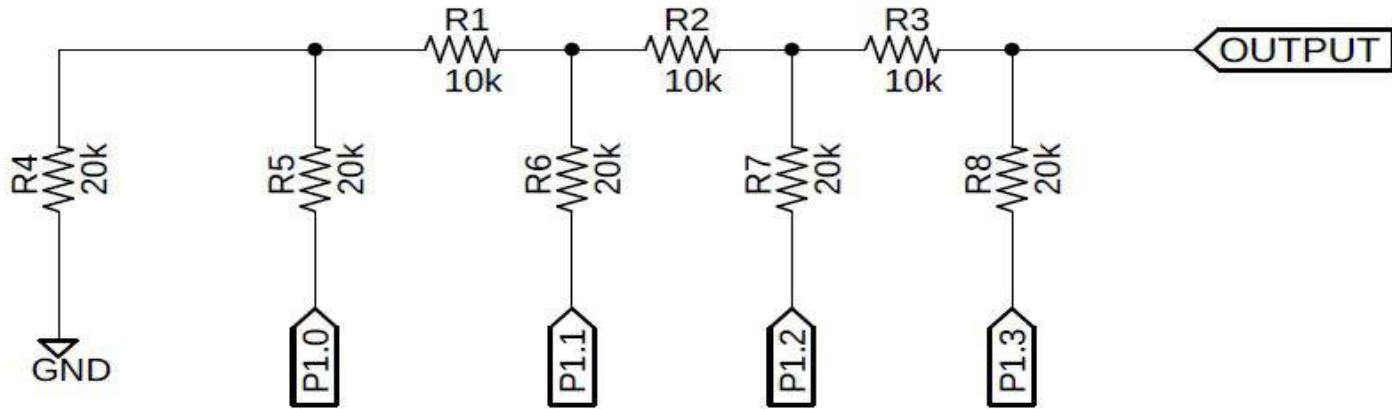
```
1 #include <msp430.h>
2 #include <inttypes.h>
3
4 char count = 0;
5 unsigned int i;
6
7 /**
8 * @brief
9 * These settings are w.r.t enabling GPIO on LunchBox
10 */
11 void register_settings_for_GPIO()
12 {
13     P1DIR |= 0xFF;                                //P1.0 to P1.7 are set as Output
14     P1OUT &= ~0xFF;                               //Initially they are set to logic zero
15 }
16
17 /*@brief entry point for the code*/
18 void main(void)
19 {
20     WDTCTL = WDTPW + WDTHOLD;                  //! Stop Watch dog (Not recommended for code in production and devices working in field)
21
22     register_settings_for_GPIO();
23
24     while(1)
25     {
26         P1OUT = count;                           // Assign value of Count to PORT 1 to represent it as binary number
27         for(i = 0; i < 5000; i++);              // Increment count
28         count++;
29     }
30 }
```



Output - 8 bit DAC



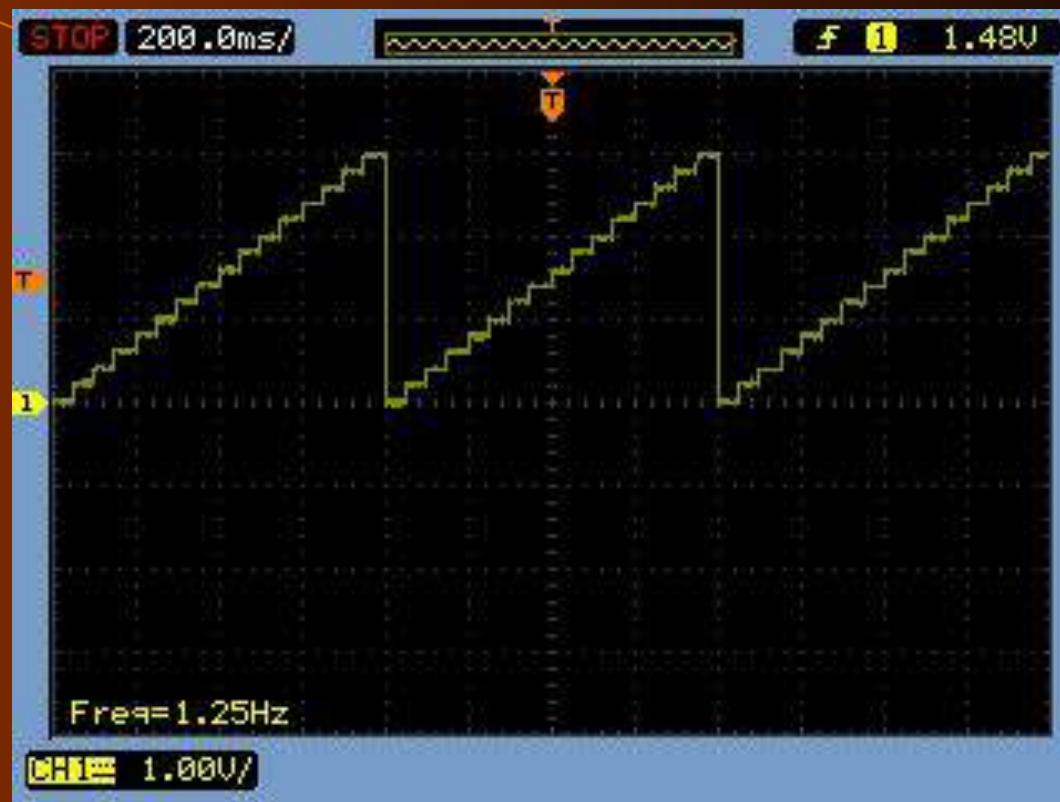
4 bit R-2R Ladder DAC



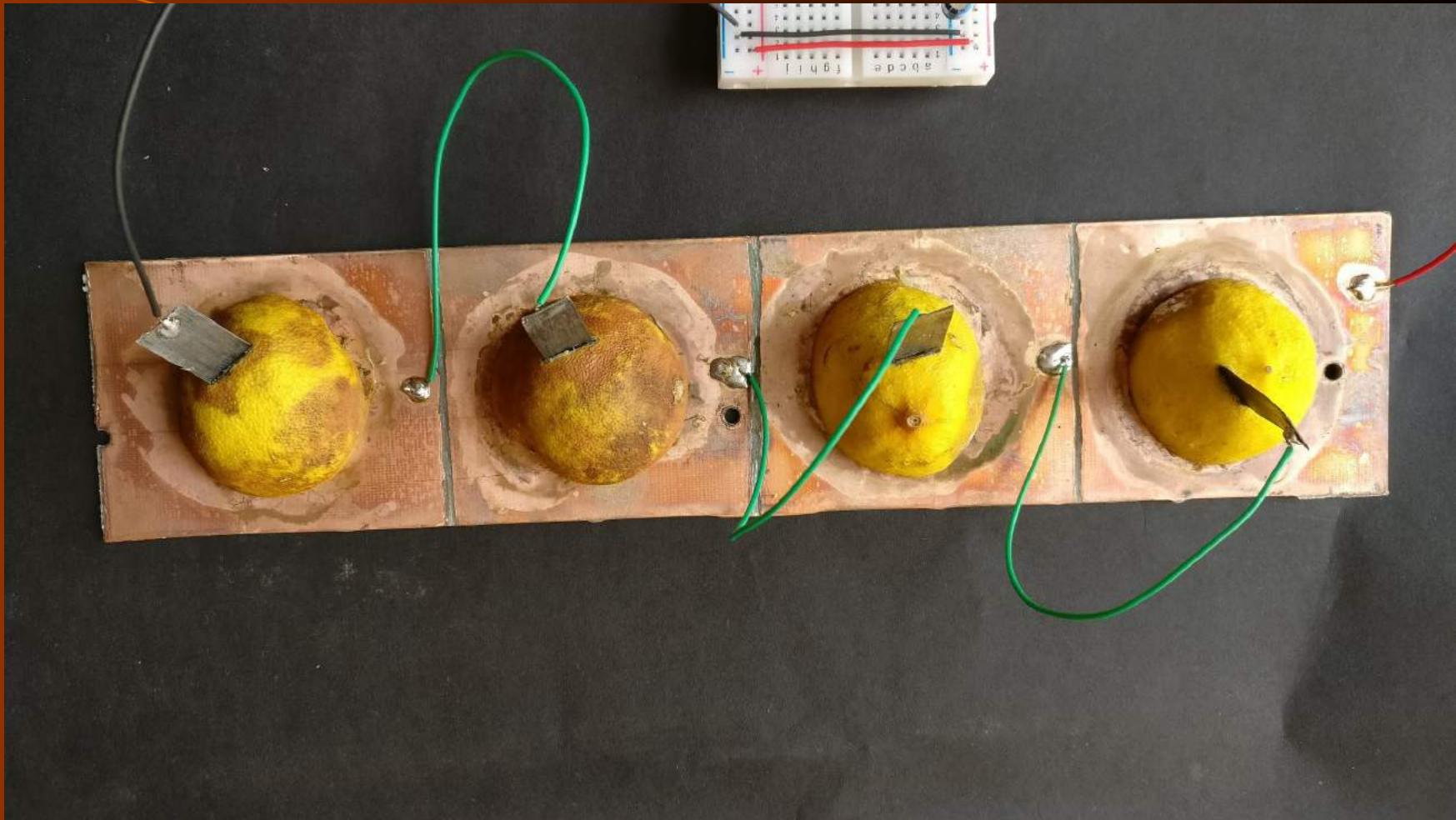
4 BIT DAC

$$V_{out} = V_{b0} / 16 + V_{b1} / 8 + V_{b2} / 4 + V_{b3} / 2$$

Output - 4 bit DAC



Lemon Battery Experiment



Lemon Battery Experiment

```
1 #include <msp430.h>
2 /**
3 * @brief
4 * These settings are wrt enabling GPIO on Lunchbox
5 */
6 void register_settings_for_GPIO()
7 {
8     P1DIR |= BIT7;                                // P1.7 as Output
9     P1OUT &= ~BIT7;                               // Initialize with zero
10}
11 /**
12 * @brief
13 * These settings are w.r.t enabling TIMER0 on Lunch Box
14 */
15 void register_settings_for_TIMER0()
16 {
17     CCTL0 = CCIE;                                // CCR0 interrupt enabled
18     TACTL = TASSEL_1 + MC_1;                      // ACLK = 32768 Hz, Up mode
19     CCR0 = 32768;                                // 1 Hz
20}
21 /*@brief entry point for the code*/
22 void main(void)
23 {
24     WDTCTL = WDTPW + WDTHOLD;                  //! Stop Watch dog (Not recommended for code in production and devices working in field)
25     unsigned int i;
26
27     do{
28         IFG1 &= ~OFIFG;                          // Clear oscillator fault flag
29         for (i = 50000; i; i--);                // Delay
30     } while (IFG1 & OFIFG);                   // Test osc fault flag
31
32     register_settings_for_TIMER0();
33     register_settings_for_GPIO();
34     _BIS_SR(LPM3_bits + GIE);                 // Enter LPM3 w/ interrupt
35}
36 /*@brief entry point for TIMER0 interrupt vector*/
37 #pragma vector= TIMER0_A0_VECTOR
38 __interrupt void Timer_A (void)
39 {
40     P1OUT |= BIT7;                            // LED HIGH
41     __delay_cycles(200);
42     P1OUT &=~ BIT7;                           // LED LOW
43}
```



Thank you!

Introduction to Embedded System Design

Serial Communication Protocols, USCI Module in MSP430

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Communication Protocols

- A Communication protocol is a set of rules and formal descriptions defined for communicating information between two or more devices.
- In other words, they describe the format and rate in which the data will be sent from sender to receiver.

Data Transmission Modes

- **Simplex**: Communication can occur only in one direction from Device A to Device B only.
- **Half Duplex**: Data transmission can occur between Device A and Device B, but only one at a time.
- **Full Duplex**: Data transmission can occur in both directions between Device A and B simultaneously.

Communication Protocols on MSP430

- UART
- SPI
- I2C

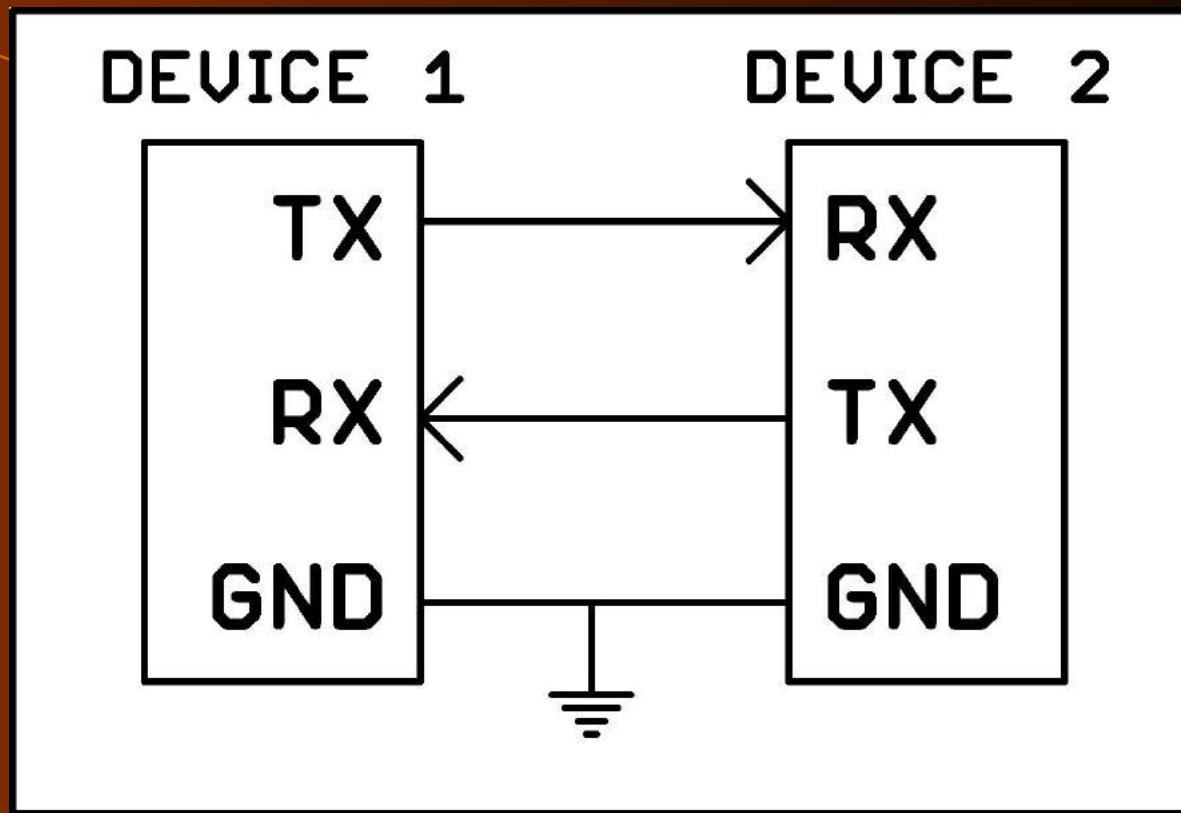
UART **(Universal Asynchronous Receiver/Transmitter)**

- Dedicated hardware meant for Serial Communication.
- Baud Rate of data transmission is known beforehand, which is defined by the user.
- UART data is organized into ‘*packets*’. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART settings), an optional *parity* bit, and 1 or 2 stop bits. Example format 9600 8N1.

List of Standard Baud Rates

- 300
- 600
- 1200
- 2400
- 4800
- 9600
- 19200
- 38400
- 57600
- 11500

UART Communication Block Diagram

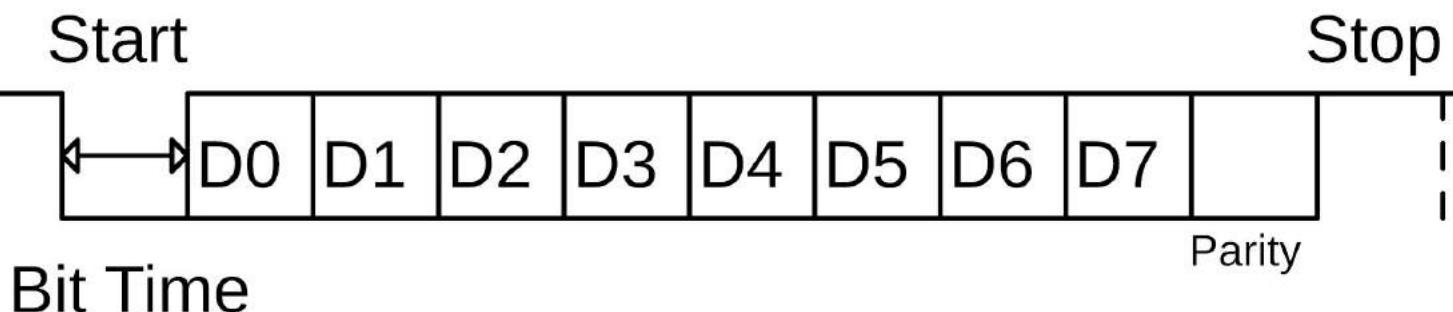


TX- Data Transmit Pin (Output Pin)

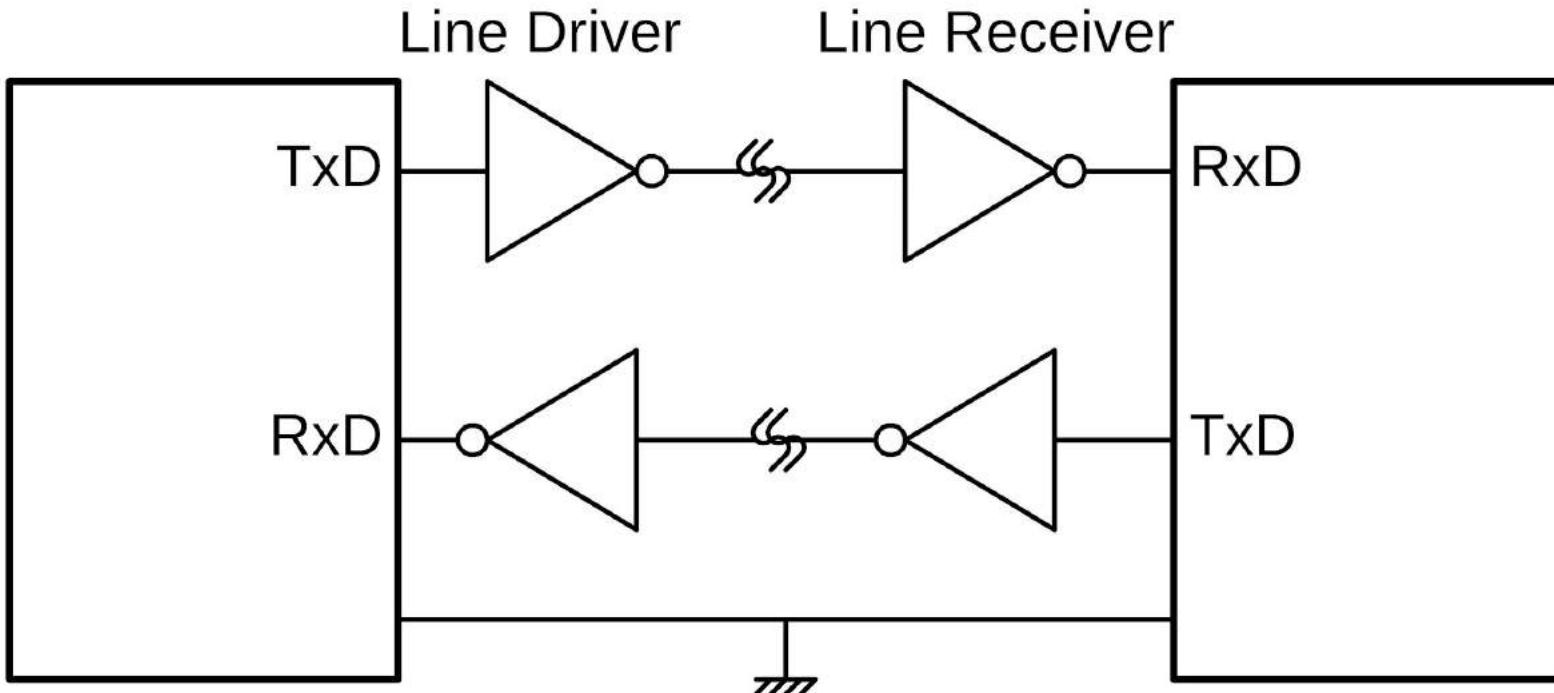
RX- Data Receive Pin (Input Pin)

GND- Reference voltage pin for the two system

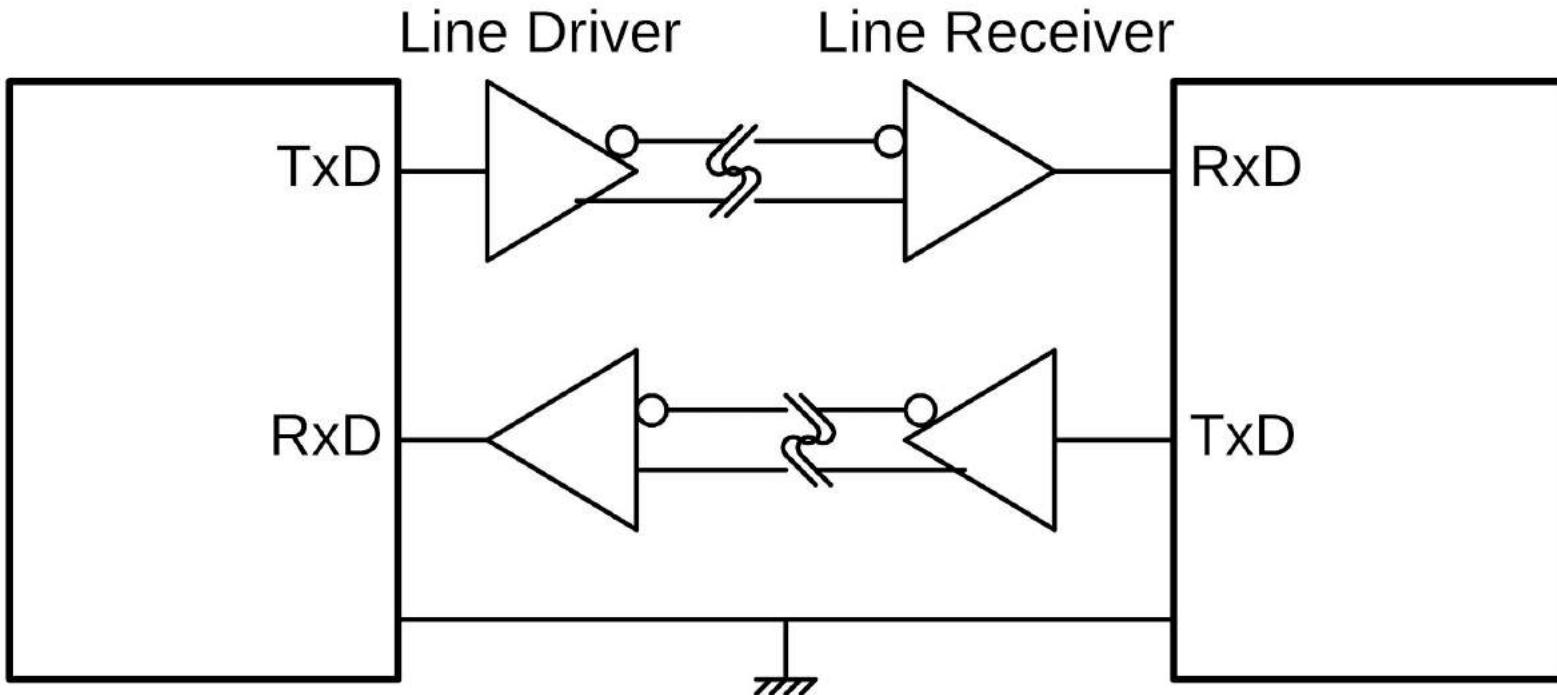
Data Format



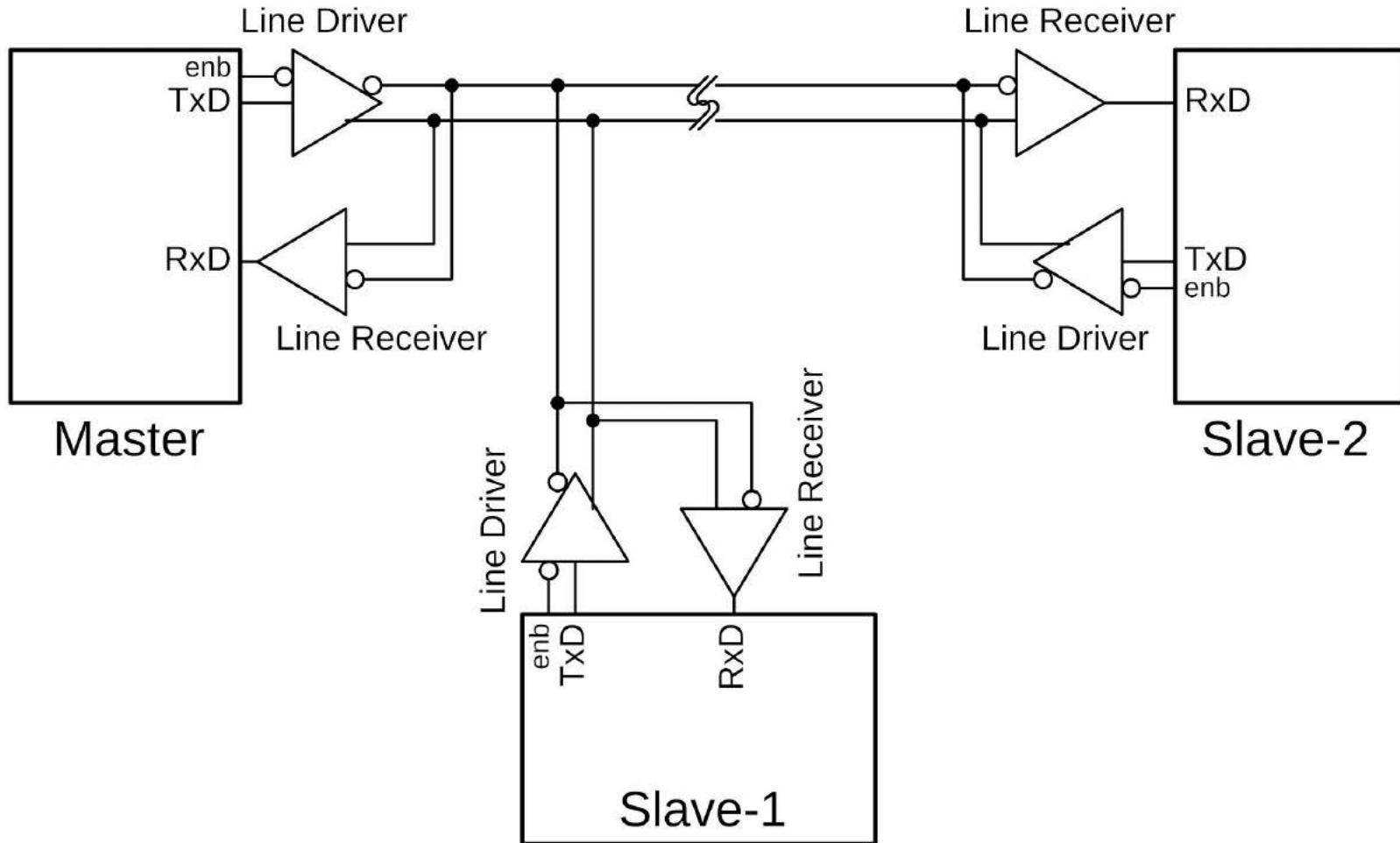
RS-232



RS-422



RS-485



Steps of Transmission

- UART module receives data to be sent from CPU in parallel form.
- UART module then adds start, stop and parity bits accordingly.
- Data is sent serially from Tx pin over the pin at predefined rate.

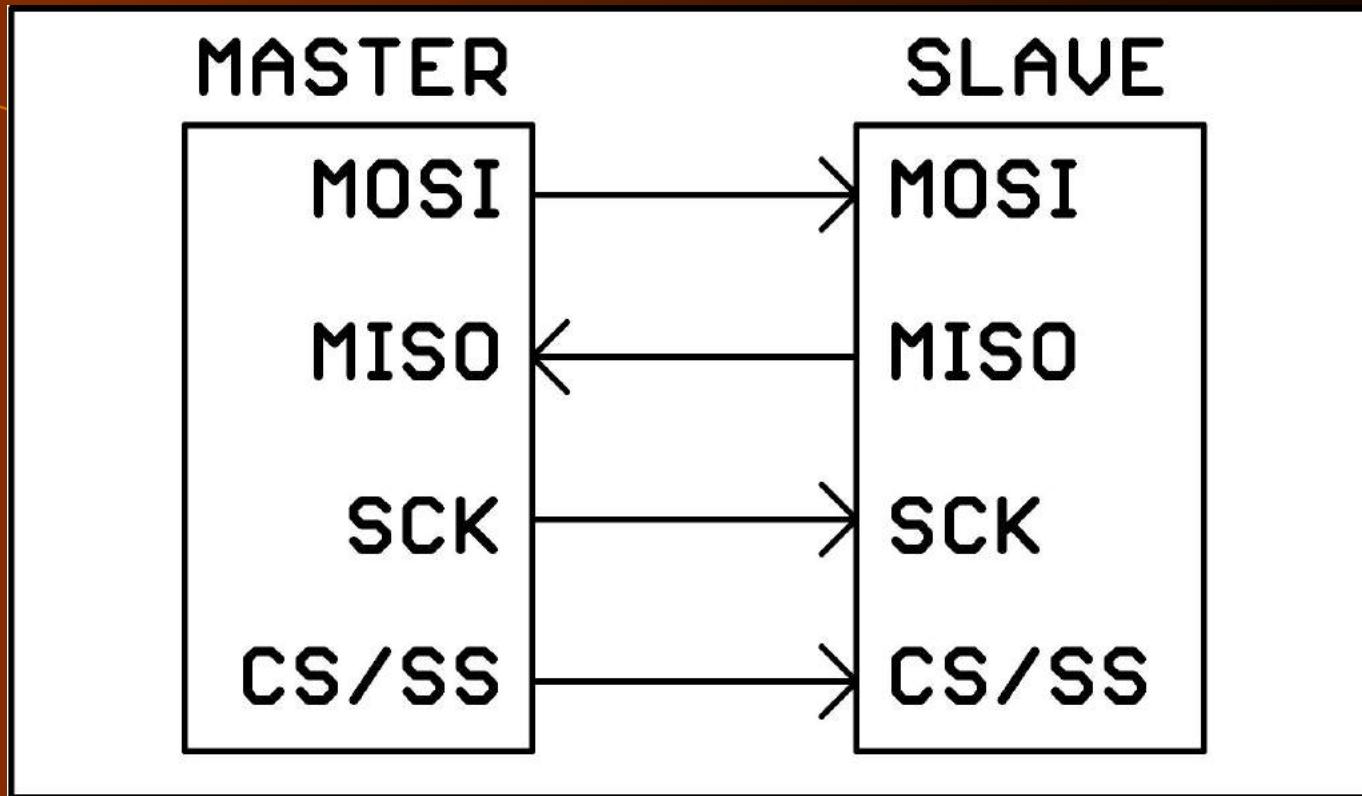
Steps of Reception

- Receiver receives data in form of packets and performs error checks using start, stop and parity bits.
- It then discards start, stop and parity bits, keeping only the data bits.
- Sends the data to CPU over data bus parallelly.

SPI (Serial Peripheral Interface)

- A Full Duplex Synchronous Serial Communication Protocol
- Master can talk to many slaves but a slave can talk to only one master

Block Diagram



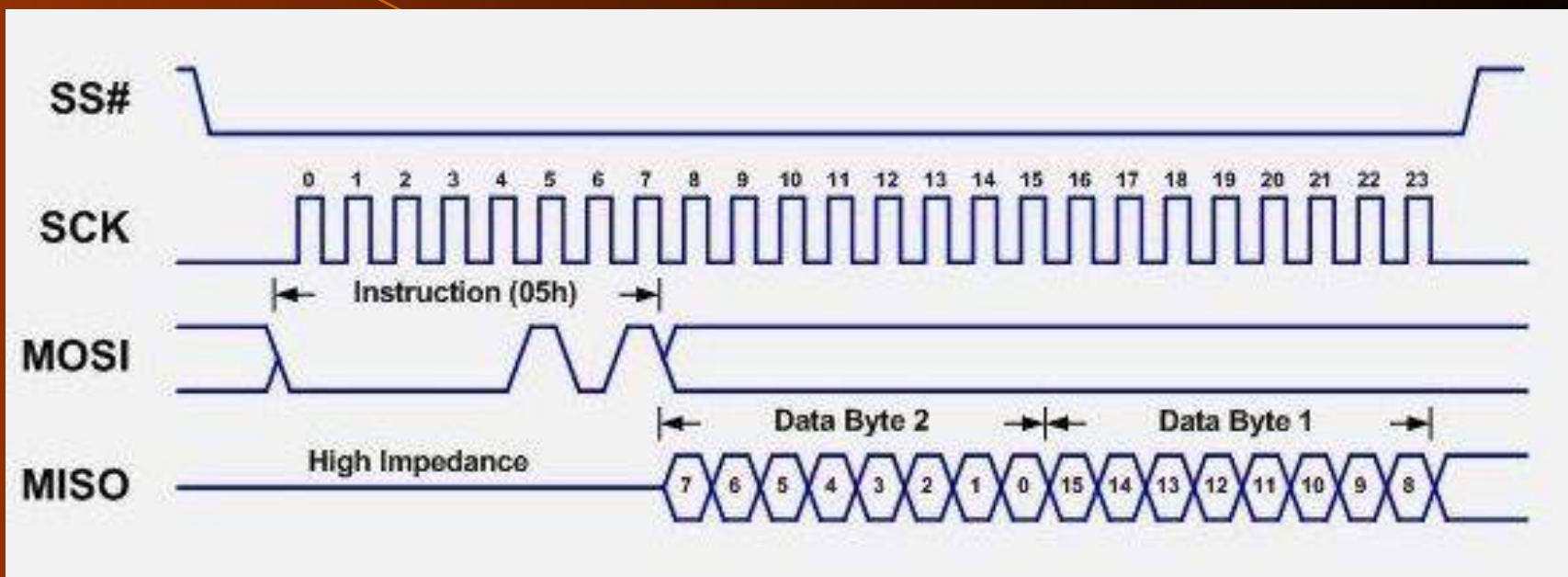
MOSI- Master Out Slave In

MISO- Master In Slave Out

SCK- Serial Clock

CS/SS- Chip Select/ Slave Select

SPI Timing Diagram



Pros and Cons

Pros:

- No start and stop bits, so data can be sent continuously.
- Fast rate due to full duplex facility.
- Multiple slaves support

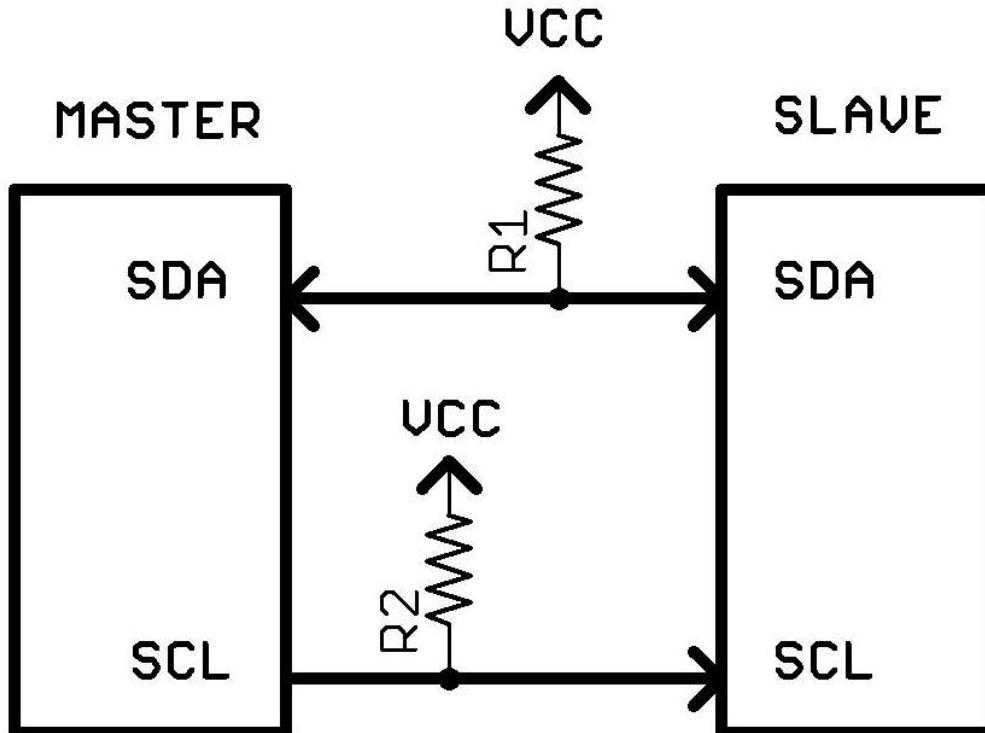
Cons:

- Cabling cost for long distance communication.
- No acknowledgement signal and error check bits.
- Master controls all the communication.
- Large number of slaves will require many separate CS/SS pins.

I²C or IIC (Inter Integrated Circuit)

- A Half Duplex Synchronous Serial Communication Protocol
- Multiple master and multiple slave support

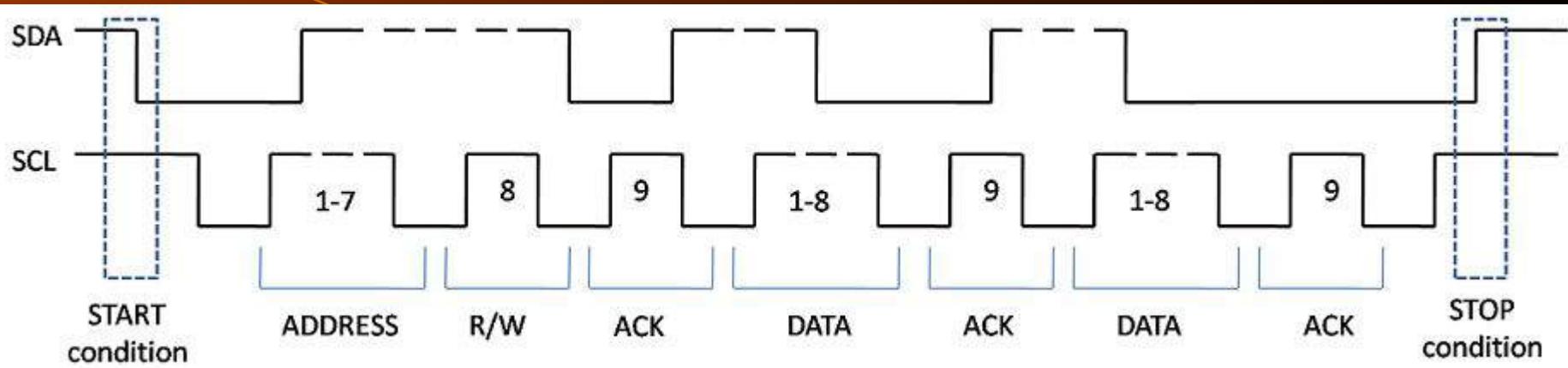
Block Diagram



SDA - Serial Data

SCL - Serial Clock

Timing Diagram



SDA - Serial Data

SCL - Serial Clock

Pros and Cons

Pros:

- Only two wires required for communication
- Multiple master and multiple slave support

Cons:

- Slower data transfer rates when compared with SPI

MSP430 Universal Serial Communication Interface

The Universal Serial Communication Interface (USCI) module in the MSP430 support multiple serial communication modes.

There are two USCI Modules in MSP430G2553:

1. The USCI_A0 module which supports:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- SPI mode

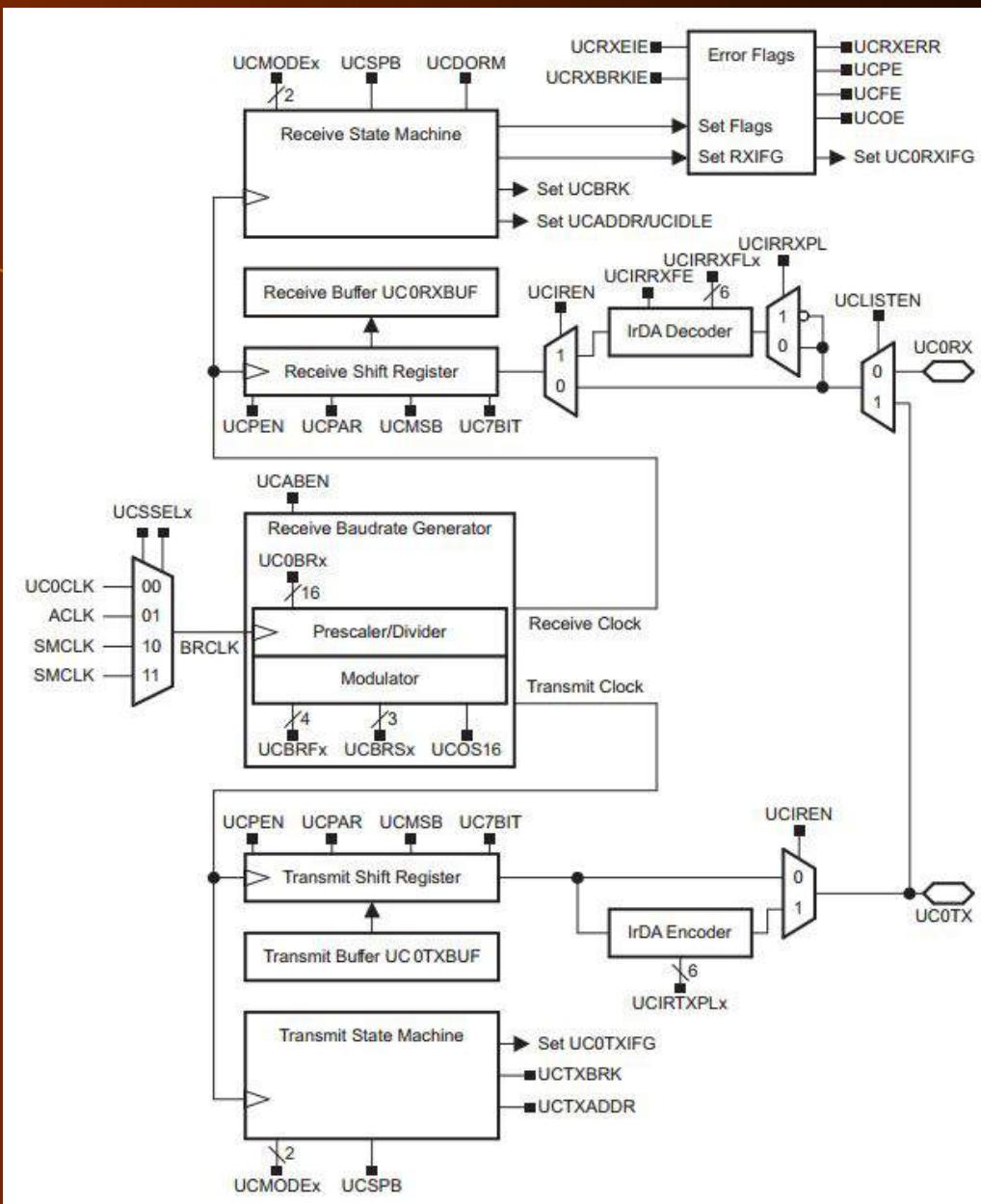
2. The USCI_B0 module which supports:

- I2C mode
- SPI mode

UART using Universal Serial Communication Interface

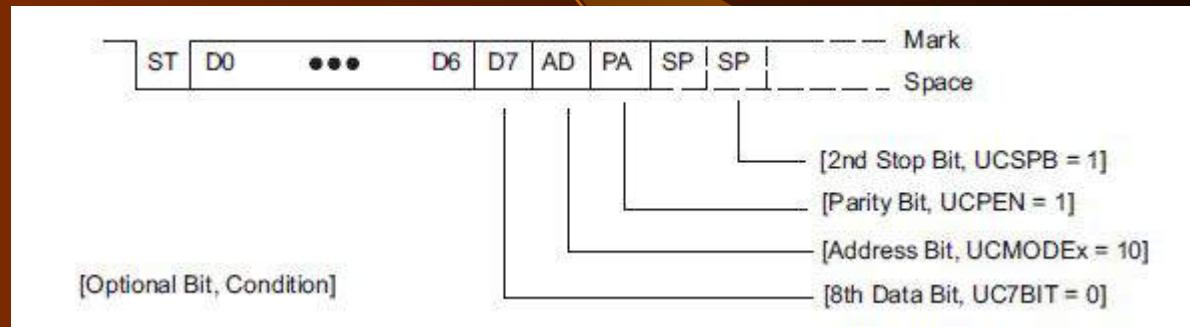
- For UART, USCI_A0 Module is used. The USCI_A0 module connects the MSP430 to an external system via two pins, UCA0RXD and UCA0TXD.
- The USCI Module can send 7 or 8-bit data with odd, even, or no parity bit
- The Module has independent transmit and receive shift registers as well as Separate transmit and receive buffer registers.
- LSB-first or MSB-first data transmit and receive
- Programmable baud rate with modulation for fractional baud rate support
- Receiver start-edge detection for auto-wake up from LPMx modes
- Status flags for error detection and suppression
- Independent interrupt capability for receive and transmit

Block diagram of UART Mode



UART Character Format

- UART Format consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first.



Setting UART Baud Rate

- UCA0BR0, UCA0BR1 and UCA0MCTL registers values are set as per the baud rate clock source and baud rate requirement.
- Values for the setting baud rate are provided in device user guide.

Example register values for baud rate

Table 15-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK Frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBRSx	UCBRFx	Maximum TX Error [%]	Maximum RX Error [%]	
32,768	1200	27	2	0	-2.8	1.4	-5.9 2.0
32,768	2400	13	6	0	-4.8	6.0	-9.7 8.3
32,768	4800	6	7	0	-12.1	5.7	-13.4 19.0
32,768	9600	3	3	0	-21.1	15.2	-44.3 21.3
1,048,576	9600	109	2	0	-0.2	0.7	-1.0 0.8
1,048,576	19200	54	5	0	-1.1	1.0	-1.5 2.5
1,048,576	38400	27	2	0	-2.8	1.4	-5.9 2.0
1,048,576	56000	18	6	0	-3.9	1.1	-4.6 5.7
1,048,576	115200	9	1	0	-1.1	10.7	-11.5 11.3
1,048,576	128000	8	1	0	-8.9	7.5	-13.8 14.8
1,048,576	256000	4	1	0	-2.3	25.4	-13.4 38.8
1,000,000	9600	104	1	0	-0.5	0.6	-0.9 1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6 0.9

Sample values, check user guide for selecting the baud rate.

USCI Tx Interrupt

- The USCI has one interrupt vector for transmission and one interrupt vector for reception.
- The UCA0TXIFG interrupt flag is set by the transmitter to indicate that UCA0TXBUF is ready to accept another character.
- An interrupt request is generated if UCA0TXIE and GIE are also set.
- UCA0TXIFG is automatically reset if a character is written to UCA0TXBUF.

USCI Rx Interrupt

- The UCA0RXIFG interrupt flag is set each time a character is received and loaded into UCA0RXBUF.
- An interrupt request is generated if UCA0RXIE and GIE are also set.
- UCA0RXIFG and UCA0RXIE are reset by a system reset PUC signal.
- UCA0RXIFG is automatically reset when UCA0RXBUF is read.

USCI Registers for UART Mode

Register	Short Form	Register Type	Address	Initial State
USCI_A0 control register 0	UCA0CTL0	Read/write	060h	Reset with PUC
USCI_A0 control register 1	UCA0CTL1	Read/write	061h	001h with PUC
USCI_A0 Baud rate control register 0	UCA0BR0	Read/write	062h	Reset with PUC
USCI_A0 baud rate control register 1	UCA0BR1	Read/write	063h	Reset with PUC
USCI_A0 modulation control register	UCA0MCTL	Read/write	064h	Reset with PUC
USCI_A0 status register	UCA0STAT	Read/write	065h	Reset with PUC
USCI_A0 receive buffer register	UCA0RXBUF	Read	066h	Reset with PUC
USCI_A0 transmit buffer register	UCA0TXBUF	Read/write	067h	Reset with PUC
USCI_A0 Auto baud control register	UCA0ABCTL	Read/write	05Dh	Reset with PUC
USCI_A0 IrDA transmit control register	UCA0IRTCTL	Read/write	05Eh	Reset with PUC
USCI_A0 IrDA receive control register	UCA0IRRCTL	Read/write	05Fh	Reset with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	00Ah with PUC

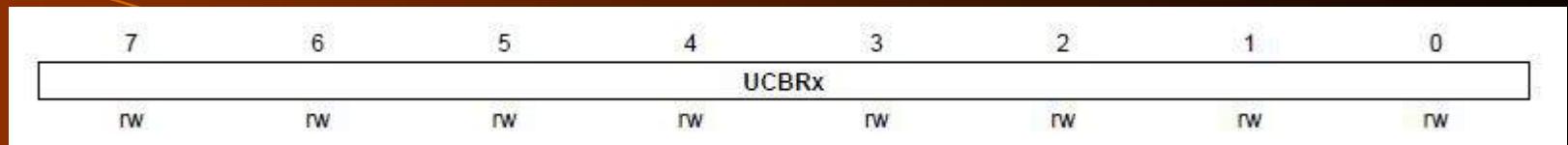
UCA0CTL0: USCI_A0 Control Register 0

	7	6	5	4	3	2	1	0
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEEx	UCSYNC	
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
UCPEN	Bit 7	Parity enable						
	0	Parity disabled.						
	1	Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.						
UCPAR	Bit 6	Parity select. UCPAR is not used when parity is disabled.						
	0	Odd parity						
	1	Even parity						
UCMSB	Bit 5	MSB first select. Controls the direction of the receive and transmit shift register.						
	0	LSB first						
	1	MSB first						
UC7BIT	Bit 4	Character length. Selects 7-bit or 8-bit character length.						
	0	8-bit data						
	1	7-bit data						
UCSPB	Bit 3	Stop bit select. Number of stop bits.						
	0	One stop bit						
	1	Two stop bits						
UCMODEEx	Bits 2-1	USCI mode. The UCMODEEx bits select the asynchronous mode when UCSYNC = 0.						
	00	UART mode						
	01	Idle-line multiprocessor mode						
	10	Address-bit multiprocessor mode						
	11	UART mode with automatic baud rate detection						
UCSYNC	Bit 0	Synchronous mode enable						
	0	Asynchronous mode						
	1	Synchronous mode						

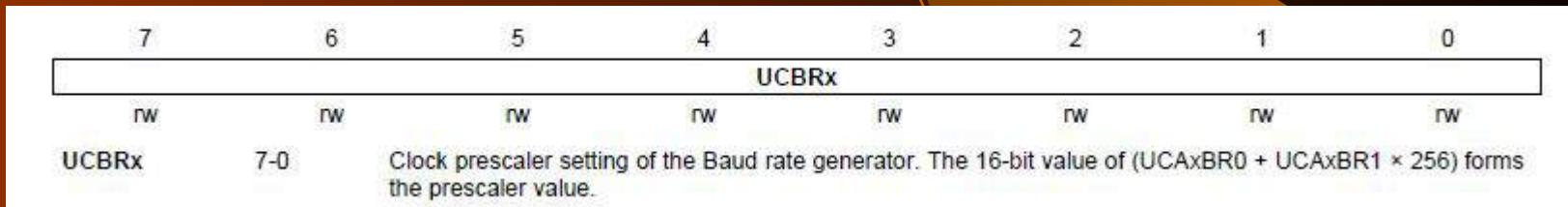
UCA0CTL1: USCI_A0 Control Register 1

	7	6	5	4	3	2	1	0
	UCSSELx	UCRXIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST	
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1
UCSSELx	Bits 7-6	USCI clock source select. These bits select the BRCLK source clock.						
	00	UCLK						
	01	ACLK						
	10	SMCLK						
	11	SMCLK						
UCRXIE	Bit 5	Receive erroneous-character interrupt-enable						
	0	Erroneous characters rejected and UCAXRXIFG is not set						
	1	Erroneous characters received will set UCAXRXIFG						
UCBRKIE	Bit 4	Receive break character interrupt-enable						
	0	Received break characters do not set UCAXRXIFG.						
	1	Received break characters set UCAXRXIFG.						
UCDORM	Bit 3	Dormant. Puts USCI into sleep mode.						
	0	Not dormant. All received characters will set UCAXRXIFG.						
	1	Dormant. Only characters that are preceded by an idle-line or with address bit set will set UCAXRXIFG. In UART mode with automatic baud rate detection only the combination of a break and synch field will set UCAXRXIFG.						
UCTXADDR	Bit 2	Transmit address. Next frame to be transmitted will be marked as address depending on the selected multiprocessor mode.						
	0	Next frame transmitted is data						
	1	Next frame transmitted is an address						
UCTXBRK	Bit 1	Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud rate detection 055h must be written into UCAXTXBUF to generate the required break/synch fields. Otherwise 0h must be written into the transmit buffer.						
	0	Next frame transmitted is not a break						
	1	Next frame transmitted is a break or a break/synch						
UCSWRST	Bit 0	Software reset enable						
	0	Disabled. USCI reset released for operation.						
	1	Enabled. USCI logic held in reset state.						

UCA0BR0: USCI_A0 Baud Rate Control Register 0



UCA0BR1: USCI_A0 Baud Rate Control Register 1



UCA0MCTL: USCI_A0 Modulation Control Register

7	6	5	4	3	2	1	0
	UCBRFx					UCBRSx	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
UCBRFx	Bits 7-4	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 15-3 shows the modulation pattern.					
UCBRSx	Bits 3-1	Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 15-2 shows the modulation pattern.					
UCOS16	Bit 0	Oversampling mode enabled					
		0	Disabled				
		1	Enabled				

UCA0STAT: USCI_A0 Status Register

	7	6	5	4	3	2	1	0
	UCLISTEN	UCFE	UCOE	UCPE	UCBRK	UCRXERR	UCADDR UCIDLE	UCBUSY
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0
UCLISTEN	Bit 7							
		Listen enable. The UCLISTEN bit selects loopback mode.						
		0	Disabled					
		1	Enabled. UCAXTXD is internally fed back to the receiver.					
UCFE	Bit 6							
		Framing error flag						
		0	No error					
		1	Character received with low stop bit					
UCOE	Bit 5							
		Overrun error flag. This bit is set when a character is transferred into UCAXRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it will not function correctly.						
		0	No error					
		1	Overrun error occurred					
UCPE	Bit 4							
		Parity error flag. When UCOPEN = 0, UCPE is read as 0.						
		0	No error					
		1	Character received with parity error					
UCBRK	Bit 3							
		Break detect flag						
		0	No break condition					
		1	Break condition occurred					
UCRXERR	Bit 2							
		Receive error flag. This bit indicates a character was received with error(s). When UCRXERR = 1, one or more error flags (UCFE, UCPE, UCOE) is also set. UCRXERR is cleared when UCAXRXBUF is read.						
		0	No receive errors detected					
		1	Receive error detected					
UCADDR	Bit 1							
		Address received in address-bit multiprocessor mode.						
		0	Received character is data					
		1	Received character is an address					
UCIDLE								
		Idle line detected in idle-line multiprocessor mode.						
		0	No idle line detected					
		1	Idle line detected					
UCBUSY	Bit 0							
		USCI busy. This bit indicates if a transmit or receive operation is in progress.						
		0	USCI inactive					
		1	USCI transmitting or receiving					

UCA0RXBUF: USCI_A0 Receive Buffer Register

UCA0TBUF: USCI_A0 Transmit Buffer Register

IE2: Interrupt Enable Register 2

IFG2: Interrupt Flag Register 2

	7	6	5	4	3	2	1	0
							UCA0TXIFG	UCA0RXIFG
							rw-1	rw-0
UCA0TXIFG	Bits 7-2	These bits may be used by other modules (see the device-specific data sheet).						
	Bit 1	USCI_A0 transmit interrupt flag. UCA0TXIFG is set when UCA0TXBUF is empty.						
		0 No interrupt pending						
		1 Interrupt pending						
UCA0RXIFG	Bit 0	USCI_A0 receive interrupt flag. UCA0RXIFG is set when UCA0RXBUF has received a complete character.						
		0 No interrupt pending						
		1 Interrupt pending						

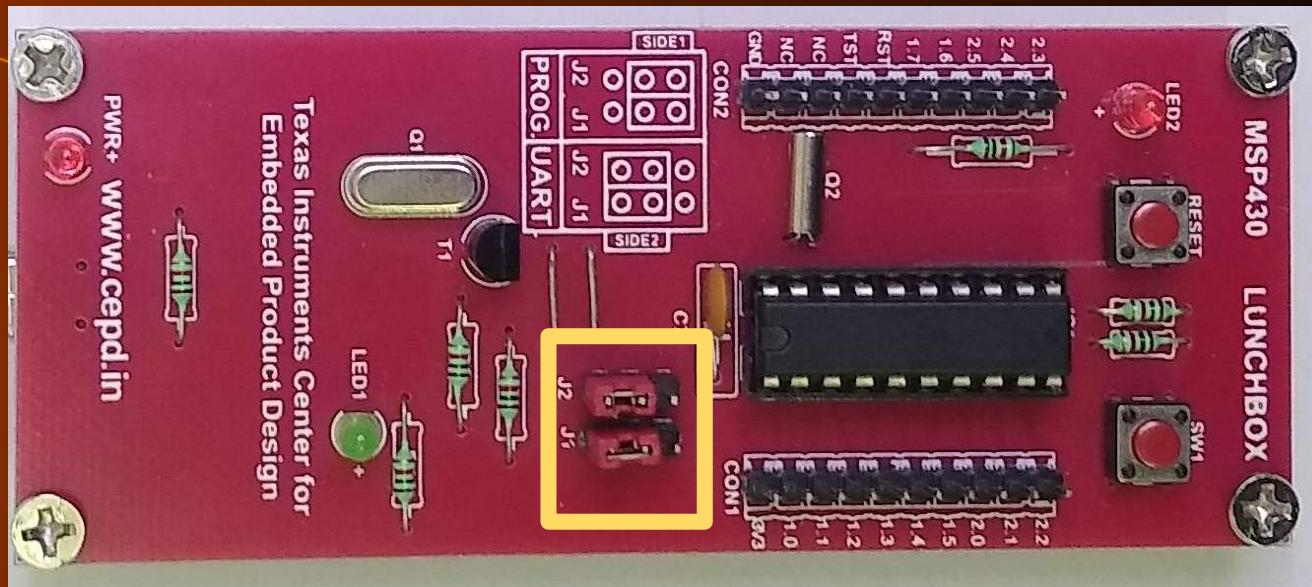
Selecting UART function on P1.1 and P1.2

P1.1/ TA0.0/ UCA0RXD/ UCA0SOMI/ A1 ⁽²⁾ / CA1/ Pin Osc	1	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.0	1	1	0	0	0
		TA0.CCI0A	0	1	0	0	0
		UCA0RXD	from USCI	1	1	0	0
		UCA0SOMI	from USCI	1	1	0	0
		A1	X	X	X	1 (y = 1)	0
		CA1	X	X	X	0	1 (y = 1)
		Capacitive sensing	X	0	1	0	0
		P1.2 (I/O)	I: 0; O: 1	0	0	0	0
P1.2/ TA0.1/ UCA0TXD/ UCA0SIMO/ A2 ⁽²⁾ / CA2/ Pin Osc	2	TA0.1	1	1	0	0	0
		TA0.CCI1A	0	1	0	0	0
		UCA0TXD	from USCI	1	1	0	0
		UCA0SIMO	from USCI	1	1	0	0
		A2	X	X	X	1 (y = 2)	0
		CA2	X	X	X	0	1 (y = 2)
		Capacitive sensing	X	0	1	0	0

Example Code : Hello Serial

```
1 #include <msp430.h>
2
3 /**
4 * @brief
5 * These settings are wrt enabling uart on Lunchbox
6 */
7 void register_settings_for_UART()
8 {
9     P1SEL = BIT1 + BIT2;           // Select UART RX/TX function on P1.1,P1.2
10    P1SEL2 = BIT1 + BIT2;
11
12    UCA0CTL1 |= UCSSEL_1;         // UART Clock -> ACLK
13    UCA0BR0 = 3;                 // Baud Rate Setting for 32kHz 9600
14    UCA0BR1 = 0;                 // Baud Rate Setting for 32kHz 9600
15    UCA0MCTL = UCBRF_0 + UCBRS_3; // Modulation Setting for 32kHz 9600
16    UCA0CTL1 &= ~UCSWRST;        // Initialize UART Module
17    IE2 |= UCA0RXIE;            // Enable RX interrupt
18 }
19
20 /*@brief entry point for the code*/
21 void main(void)
22 {
23     WDTCTL = WDTPW + WDTHOLD;    //! Stop Watchdog (Not recommended for code in production and devices working in field)
24
25     register_settings_for_UART();
26
27     __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, Enable Interrupt
28
29 }
30
31 /**
32 * @brief
33 * Interrupt Vector for UART on LunchBox
34 */
35 #pragma vector=USCIAB0RX_VECTOR      // UART RX Interrupt Vector
36 _interrupt void USCI0RX_ISR(void)
37 {
38     while (!(IFG2&UCA0TXIFG));       // Check if TX is ongoing
39     UCA0TXBUF = UCA0RXBUF + 1;       // TX -> Received Char + 1
40 }
```

Lunchbox Jumper Settings for UART

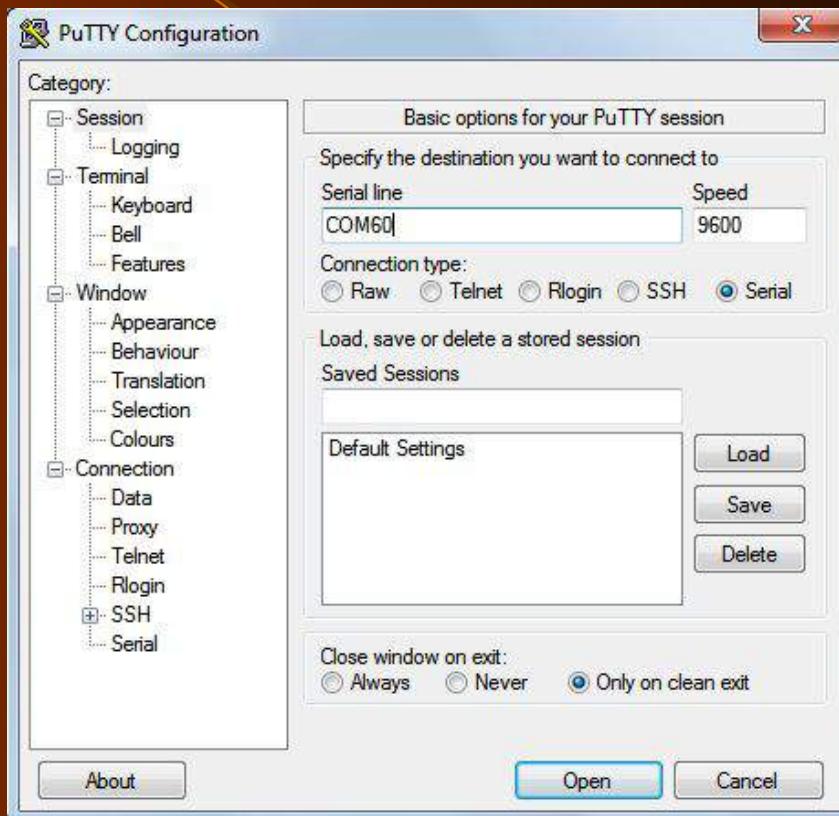


Jumper position needs to be changed once the code has been uploaded.

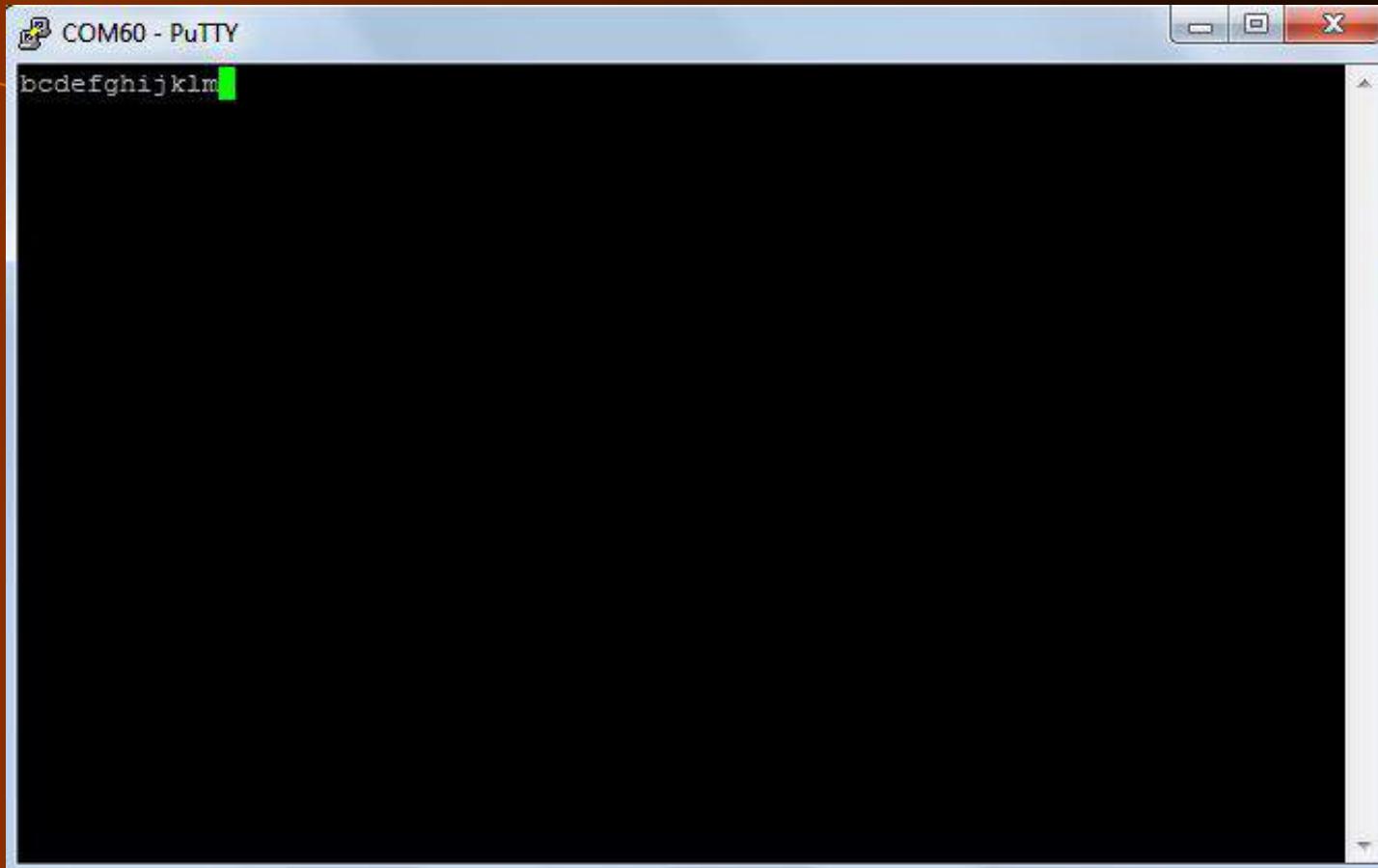
Download and Install PuTTY

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application.

PuTTY can be used as a serial terminal for this Code Example.

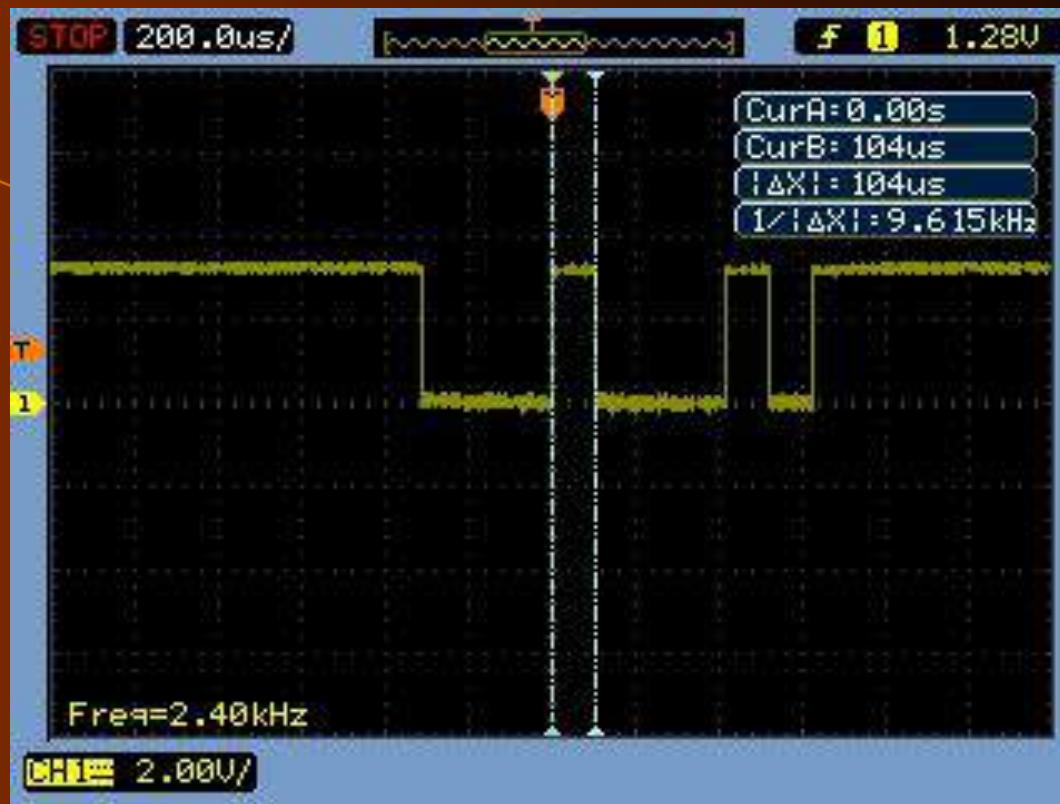


Output on PuTTY



Receive = Transmit + 1 (as per ASCII value)

a → b, b → c A → B, B → C 1 → 2, 2 → 3



Oscilloscope output at Rx pin of microcontroller when 'D' is received.



Thank you!

Introduction to Embedded System Design

MSP430 Timer in Capture Mode

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

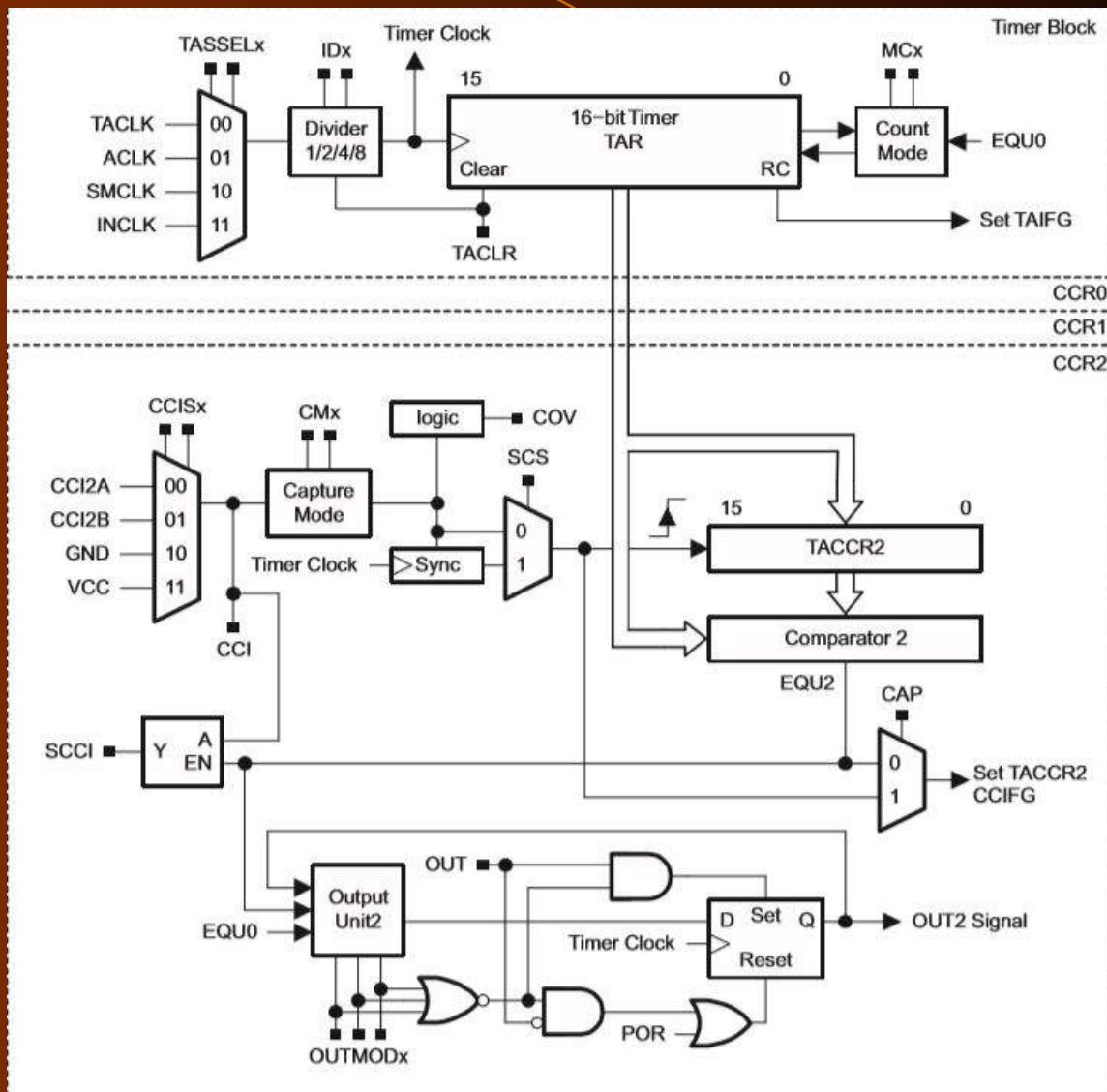
Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

TIMER_A Block Diagram



TIMER_A Registers

1. Timer_A control - TACTL
2. Timer_A counter - TAR
3. Timer_A capture/compare control 0 - TACCTL0
4. Timer_A capture/compare 0 - TACCR0
5. Timer_A capture/compare control 1 - TACCTL1
6. Timer_A capture/compare 1 - TACCR1
7. Timer_A capture/compare control 2 - TACCTL2
8. Timer_A capture/compare 2 - TACCR2
9. Timer_A Interrupt vector - TAIV

TACTL, Timer_A Control Register

TACCTL Continued

TASSELx	Bits 9-8	Timer_A clock source select
	00	TACLK
	01	ACLK
	10	SMCLK
	11	INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock.
	00	/1
	01	/2
	10	/4
	11	/8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.
	00	Stop mode: the timer is halted.
	01	Up mode: the timer counts up to TACCR0.
	10	Continuous mode: the timer counts up to 0FFFFh.
	11	Up/down mode: the timer counts up to TACCR0 then down to 0000h.
Unused	Bit 3	Unused
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request.
	0	Interrupt disabled
	1	Interrupt enabled
TAIFG	Bit 0	Timer_A interrupt flag
	0	No interrupt pending
	1	Interrupt pending

TACCTL_x, Capture/Compare Control Register

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx		CCIE		CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

TACCTLx Continued

CMx	Bit 15-14	Capture mode
	00	No capture
	01	Capture on rising edge
	10	Capture on falling edge
	11	Capture on both rising and falling edges
CCISx	Bit 13-12	Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections.
	00	CCIxA
	01	CCIxB
	10	GND
	11	V _{cc}
SCS	Bit 11	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
	0	Asynchronous capture
	1	Synchronous capture
SCCI	Bit 10	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit
Unused	Bit 9	Unused. Read only. Always read as 0.
CAP	Bit 8	Capture mode
	0	Compare mode
	1	Capture mode

TACCTLx Continued

OUTMODx	Bits 7-5	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0.
	000	OUT bit value
	001	Set
	010	Toggle/reset
	011	Set/reset
	100	Toggle
	101	Reset
	110	Toggle/set
	111	Reset/set
CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.
	0	Interrupt disabled
	1	Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
OUT	Bit 2	Output. For output mode 0, this bit directly controls the state of the output.
	0	Output low
	1	Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
	0	No capture overflow occurred
	1	Capture overflow occurred
CCIFG	Bit 0	Capture/compare interrupt flag
	0	No interrupt pending
	1	Interrupt pending

TAIV, Timer_A Interrupt Vector Register

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	0	0	0		TAIVx		0
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TAIVx Bits 15-0 Timer_A interrupt vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	-	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2 ⁽¹⁾	TACCR2 CCIFG	
06h	Reserved	-	
08h	Reserved	-	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	-	
0Eh	Reserved	-	Lowest

⁽¹⁾ Not implemented in MSP430x20xx devices

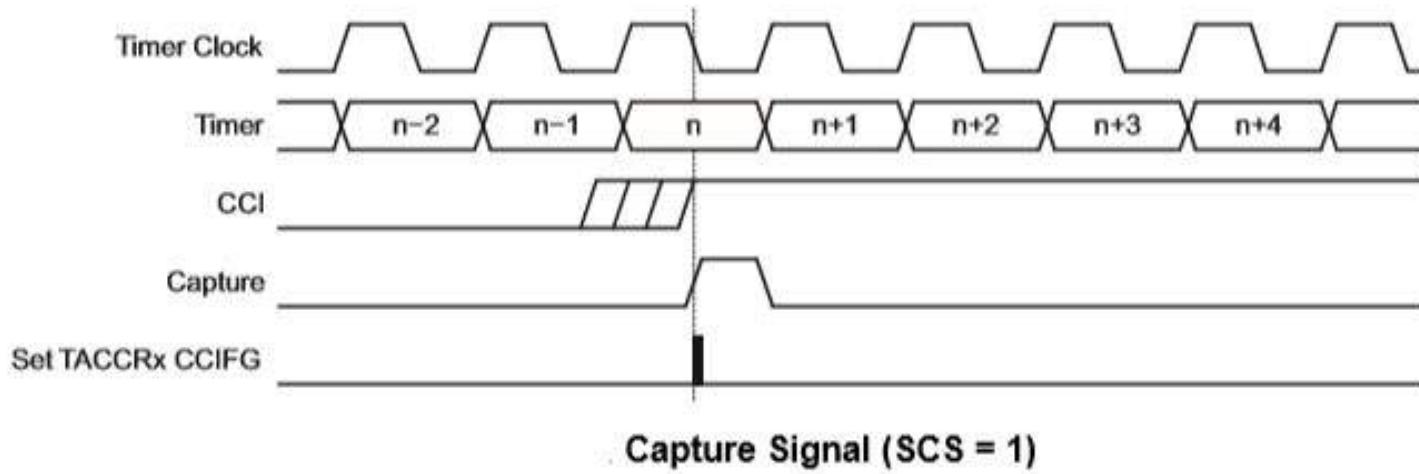
Timer Capture Mode

- The capture mode of timer is selected when CAP = 1 in TACCTLx register.
- CCISx bits decide the capture input. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals.
- When an event occurs at input signal, the value in TAR register is stored in TACCRx register.
- The CMx bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal.
- If a capture occurs:
 - The timer value is copied into the TACCRx register
 - The interrupt flag CCIFG is set in TACCTLx register

Timer Capture Mode

- State of selected input can be read in the CCI bit of TACCTLn.
- This input is asynchronous and can cause hardware “race condition”.
- The hardware of capture mode includes a synchronizer which can be enabled by setting the SCS bit.
- Capture events are then synchronised with the next falling edge of the timer clock that follows the trigger, midway between increments of TAR.
- SCS bit should normally be set for safety

Timer Capture Mode



Timer Capture Mode

- Many applications compute difference in time between two successive captures.
- First capture value must be stored away so that TACCRn is ready for the next capture.
- Capture overflow bit COV is set if second capture is performed before the value from first capture is read.
- This warns the program that an event has been missed

Frequency Measurement Example Code

1. Period Measurement.
2. Direct Frequency Measurement.

Period Measurement (Timer Capture)

Measure the period by counting the edges of a known source frequency signal between the two consecutive rising/falling edges of the unknown frequency.

Reference signal → ACKL/4 (Timer0)

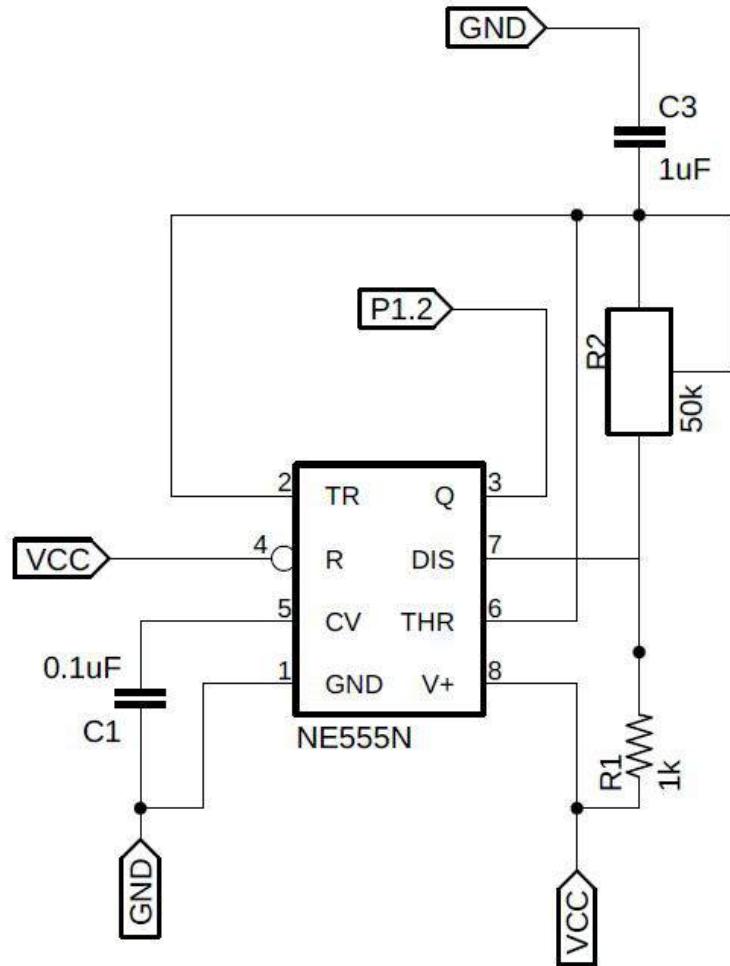
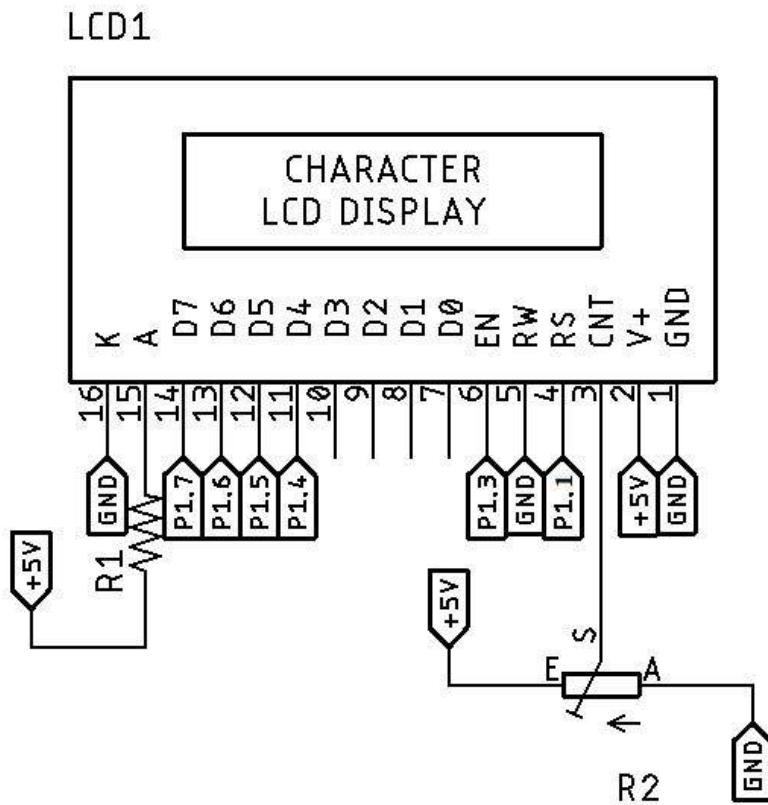
Unknown Signal → TA0.CCI1A Timer0 capture (TA0.1)

Reference Signal



Unknown Signal

Period Measurement Test Setup



Selecting Timer Function on P1.2

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ^[1]				
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x=1 ^[2]	CAPD.y
P1.0/ TA0CLK/ ACLK/ A0 ⁽²⁾ / CA0/ Pin Osc	0	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.TACLK	0	1	0	0	0
		ACLK	1	1	0	0	0
		A0	X	X	X	1 (y = 0)	0
		CA0	X	X	X	0	1 (y = 0)
		Capacitive sensing	X	0	1	0	0
P1.1/ TA0.0/ UCA0RXD/ UCA0SOMI/ A1 ⁽²⁾ / CA1/ Pin Osc	1	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.0	1	1	0	0	0
		TA0.CCI0A	0	1	0	0	0
		UCA0RXD	from USCI	1	1	0	0
		UCA0SOMI	from USCI	1	1	0	0
		A1	X	X	X	1 (y = 1)	0
		CA1	X	X	X	0	1 (y = 1)
		Capacitive sensing	X	0	1	0	0
P1.2/ TA0.1/ UCA0TXD/ UCA0SIMO/ A2 ⁽²⁾ / CA2/ Pin Osc	2	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.1	1	1	0	0	0
		TA0.CCI1A	0	1	0	0	0
		UCA0TXD	from USCI	1	1	0	0
		UCA0SIMO	from USCI	1	1	0	0
		A2	X	X	X	1 (y = 2)	0
		CA2	X	X	X	0	1 (y = 2)
		Capacitive sensing	X	0	1	0	0

```
1 #include <msp430.h>
2 #include <inttypes.h>
3 #include <stdio.h>
4
5 #define CMD      0
6 #define DATA     1
7
8 #define LCD_OUT   P1OUT
9 #define LCD_DIR   P1DIR
10 #define D4       BIT4
11 #define D5       BIT5
12 #define D6       BIT6
13 #define D7       BIT7
14 #define RS      BIT1
15 #define EN      BIT3
16
17 volatile unsigned char count;
18 volatile unsigned int edge1, edge2, period; // Global variables
19 volatile double freq, time;
20
21
22 /**
23 * @brief Delay function for producing delay in 0.1 ms increments
24 * @param t milliseconds to be delayed
25 * @return void
26 */
27 void delay(uint16_t t)
28 {
29     uint16_t i;
30     for(i=t; i > 0; i--)
31         __delay_cycles(100);
32 }
33
```

```
32
33 /**
34 * @brief Function to pulse EN pin after data is written
35 * @return void
36 */
37 void pulseEN(void)
38 {
39     LCD_OUT |= EN;
40     delay(1);
41     LCD_OUT &= ~EN;
42     delay(1);
43 }
44
45 /**
46 * @brief Function to write data/command to LCD
47 * @param value Value to be written to LED
48 * @param mode Mode -> Command or Data
49 * @return void
50 */
51 void lcd_write(uint8_t value, uint8_t mode)
52 {
53     if(mode == CMD)
54         LCD_OUT &= ~RS;                      // Set RS -> LOW for Command mode
55     else
56         LCD_OUT |= RS;                     // Set RS -> HIGH for Data mode
57
58     LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));           // Write high nibble first
59     pulseEN();
60     delay(1);
61
62     LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));    // Write low nibble next
63     pulseEN();
64     delay(1);
65 }
66
```

```
66
67 /**
68 * @brief Function to print a string on LCD
69 * @param *s pointer to the character to be written.
70 * @return void
71 */
72 void lcd_print(char *s)
73 {
74     while(*s)
75     {
76         lcd_write(*s, DATA);
77         s++;
78     }
79 }
80
81 /**
82 * @brief Function to convert number into character array
83 * @param num integer number to be converted.
84 * @return void
85 */
86 void lcd_printNumber(unsigned int num)
87 {
88     char buf[6];
89     char *str = &buf[5];
90
91     *str = '\0';
92
93     do
94     {
95         unsigned long m = num;
96         num /= 10;
97         char c = (m - 10 * num) + '0';
98         *--str = c;
99     } while(num);
100
101    lcd_print(str);
102}
103
```

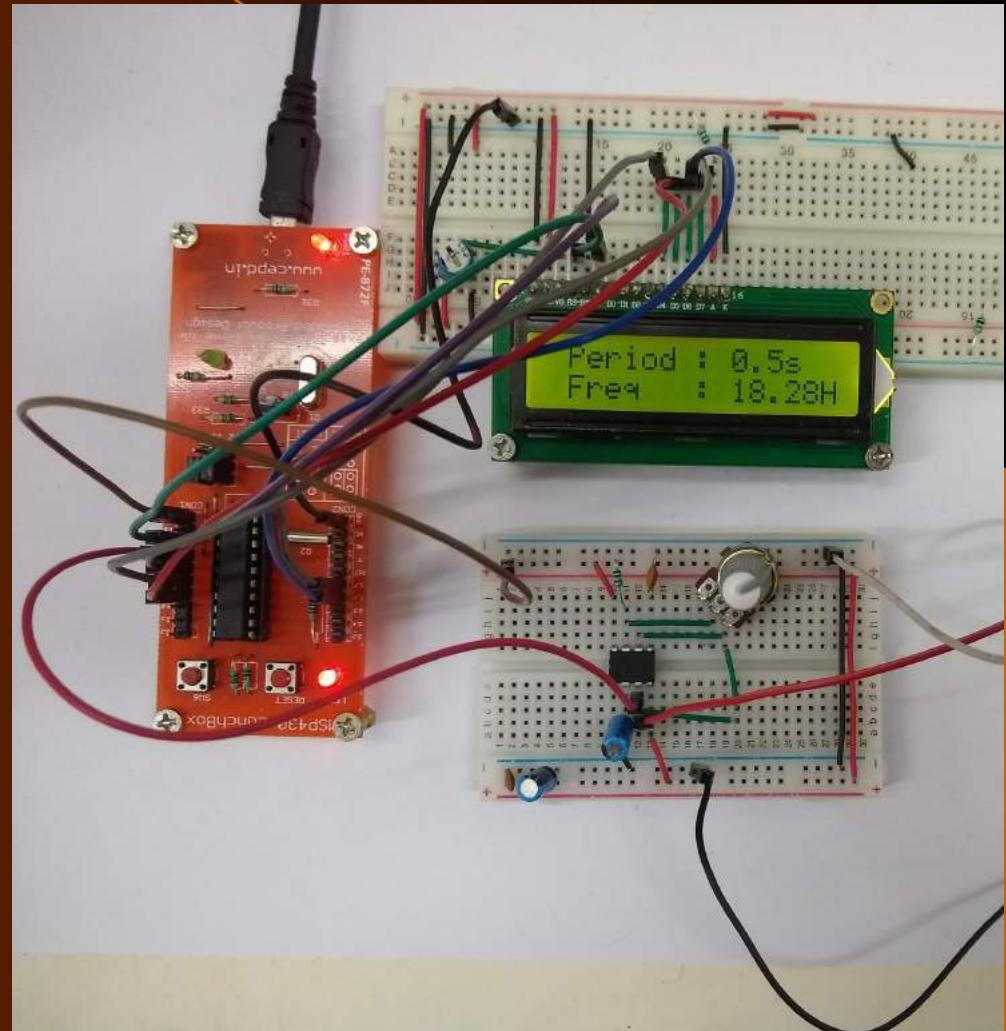
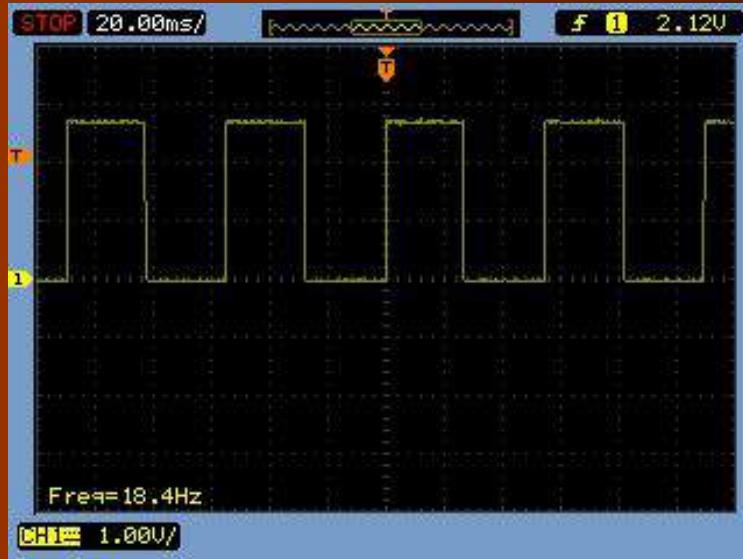
```
104 /**
105 * @brief Function to move cursor to desired position on LCD
106 * @param row Row Cursor of the LCD
107 * @param col Column Cursor of the LCD
108 * @return void
109 */
110 void lcd_setCursor(uint8_t row, uint8_t col)
111 {
112     const uint8_t row_offsets[] = { 0x00, 0x40};
113     lcd_write(0x80 | (col + row_offsets[row]), CMD);
114     delay(1);
115 }
116 /**
117 * @brief Initialize LCD
118 */
119 void lcd_init()
120 {
121     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
122     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
123
124     delay(150);                      // Wait for power up ( 15ms )
125     lcd_write(0x33, CMD);            // Initialization Sequence 1
126     delay(50);                      // Wait ( 4.1 ms )
127     lcd_write(0x32, CMD);            // Initialization Sequence 2
128     delay(1);                       // Wait ( 100 us )
129
130     // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
131
132     lcd_write(0x28, CMD);           // 4 bit mode, 2 line
133     delay(1);
134
135     lcd_write(0x0C, CMD);           // Display ON, Cursor OFF, Blink OFF
136     delay(1);
137
138     lcd_write(0x01, CMD);           // Clear screen
139     delay(20);
140
141     lcd_write(0x06, CMD);           // Auto Increment Cursor
142     delay(1);
143
144     lcd_setCursor(0,0);             // Goto Row 1 Column 1
145 }
```

```
148 /**
149 * @brief Displays on LCD
150 */
151 void lcd_display()
152 {
153     int int_part_time = time;                                // Integer part of calculated time
154     int decimal_part_time = (time - (float)int_part_time) * 100.0; // Decimal part of calculated time
155
156     int int_part_freq = freq;                                // Integer part of calculated frequency
157     int decimal_part_freq = (freq - (float)int_part_freq) * 100.0; // Decimal part of calculated frequency
158
159     lcd_write(0x01, CMD);                                     // Clear screen
160     delay(20);
161     lcd_setCursor(0,1);
162     lcd_print("Period : ");
163     lcd_printNumber(int_part_time);
164     lcd_print(".");
165     lcd_printNumber(decimal_part_time);
166     lcd_print("s");
167     lcd_setCursor(1,1);
168     lcd_print("Freq : ");
169     lcd_printNumber(int_part_freq);
170     lcd_print(".");
171     lcd_printNumber(decimal_part_freq);
172     lcd_print("Hz");
173     delay(10000);
174 }
175
176 /**
177 * @brief
178 * These settings are wrt enabling TIMER0 on Lunchbox
179 */
180 void register_settings_for_TIMER0()
181 {
182     P1DIR &= ~BIT2;                                         // Set P1.2 -> Input
183     P1SEL |= BIT2;                                         // Set P1.2 -> TA0.1 Capture Mode
184
185     TA0CCTL1 = CAP + CM_1 + CCIE + SCS + CCIS_0;          // Capture Mode, Rising Edge, Interrupt
186                                         // Enable, Synchronize, Source -> CCI0A
187     TA0CTL |= TASSEL_1 + MC_2 + TACLR + ID_2;             // Clock -> ACLK/4, Cont. Mode, Clear Timer
188 }
189
```

```
189
190 /*@brief entry point for the code*/
191 void main(void)
192 {
193     WDTCTL = WDTPW + WDTHOLD;           //! Stop Watchdog (Not recommended for code in production and devices working in field)
194
195     lcd_init();                         // Initialize LCD
196
197     register_settings_for_TIMER0();      //Initialize Timer0
198
199     while(1)
200     {
201         lcd_display();
202         __bis_SR_register(LPM0_bits + GIE);    // Enter LPM0, Enable Interrupt
203
204         //Exits LPM0 after 2 rising edges are captured
205         if(edge2 > edge1)
206         {
207             period = edge2 - edge1;            // Calculate Period
208             freq = (32768.0/period) / 4;
209             time = (period / 32768.0) * 4;
210         }
211     }
212 }
```

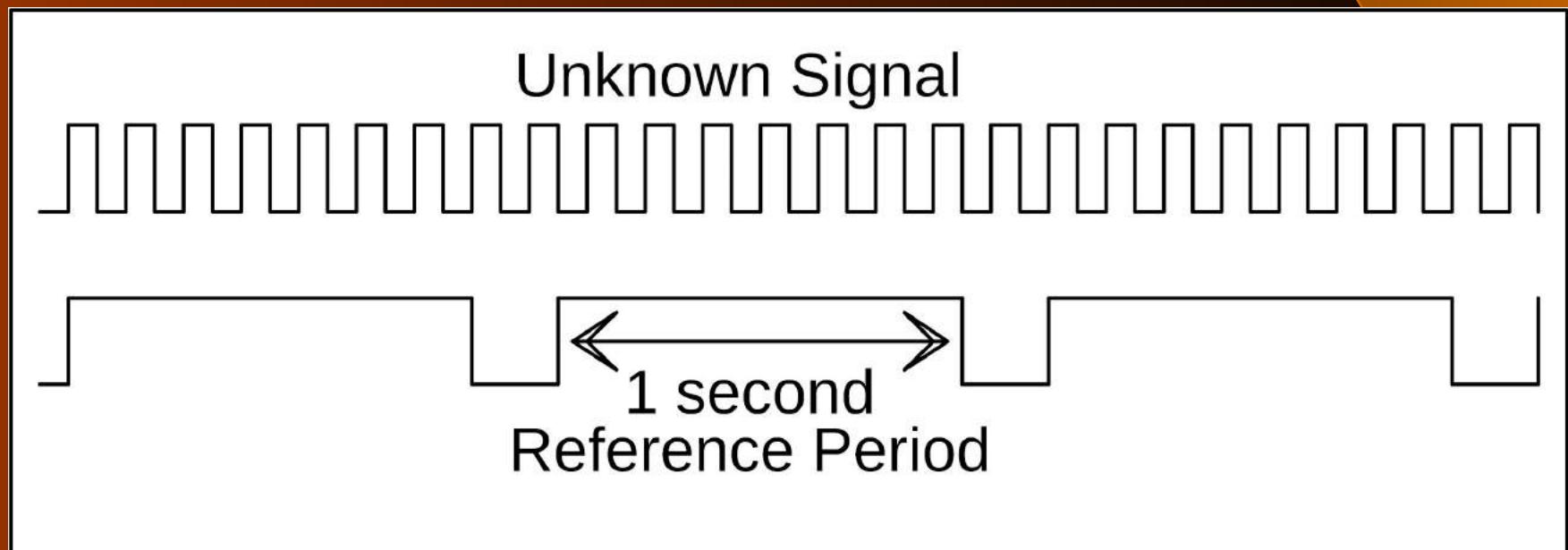
```
216
217 #pragma vector = TIMER0_A1_VECTOR
218 __interrupt void TIMER0_A1_ISR (void)
219 {
220     switch(__even_in_range(TA0IV,0x0A))
221     {
222         case TA0IV_NONE: break;                                // Vector  0: No interrupt
223
224         case TA0IV_TACCR1:                                   // Vector  2: TACCR1 CCIFG
225
226             if (count == 0)                                    // Check value of count
227             {
228                 edge1 = TA0CCR1;                            // Store timer value of 1st edge
229                 count++;                                // Increment count
230             }
231             else
232             {
233                 edge2 = TA0CCR1;                            // Store timer value of 2nd edge
234                 count=0;                                // Reset count
235                 __bic_SR_register_on_exit(LPM0_bits + GIE); // Exit LPM0 on return to main
236             }
237             break;
238
239         case TA0IV_TACCR2: break;                            // Vector  4: TACCR2 CCIFG
240         case TA0IV_6: break;                               // Vector  6: Reserved CCIFG
241         case TA0IV_8: break;                               // Vector  8: Reserved CCIFG
242         case TA0IV_TAIFG: break;                           // Vector 10: TAIFG
243         default:   break;
244     }
245 }
```

Period Measurement Test Setup



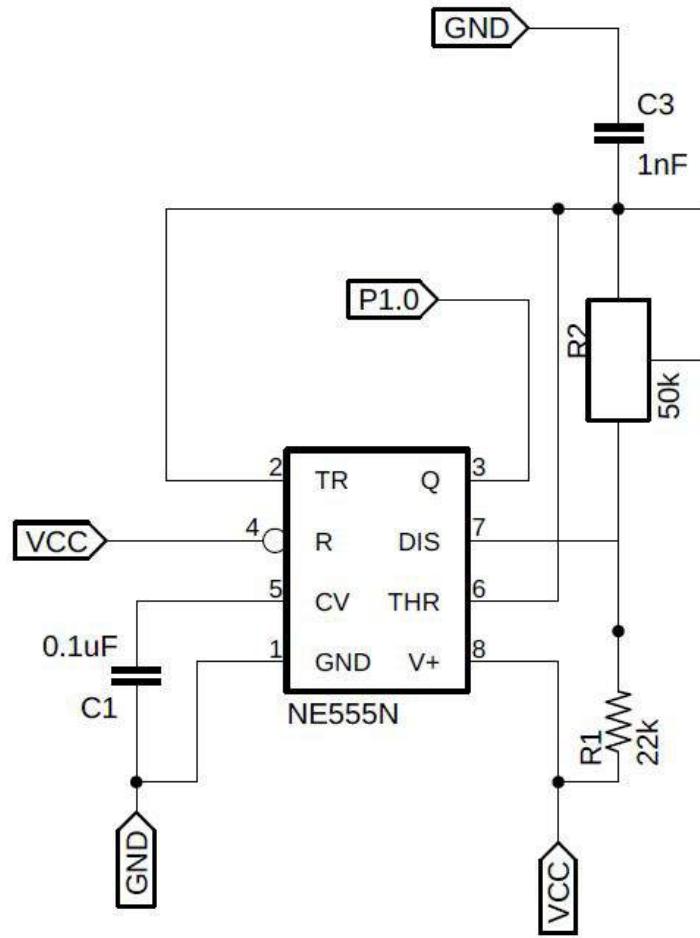
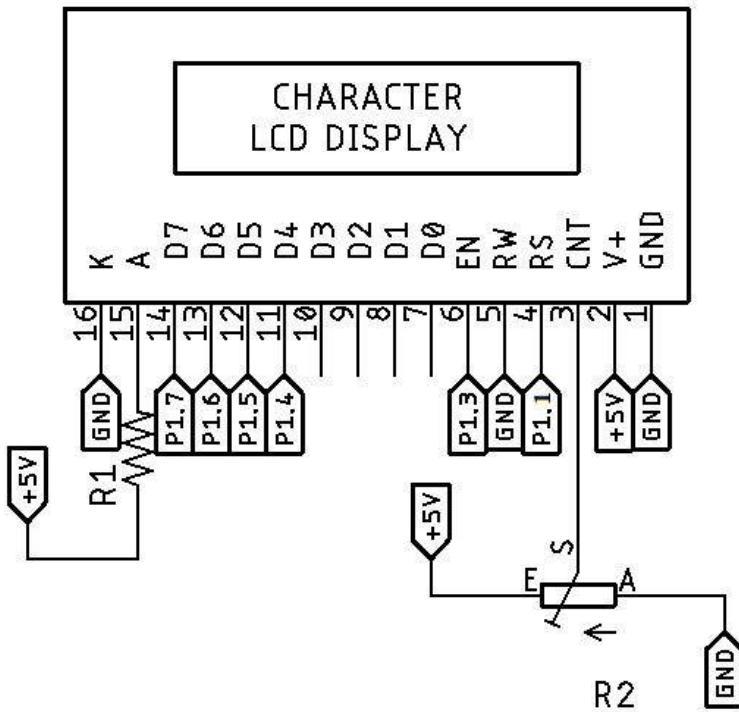
Direct Frequency Measurement (Timer as counter)

- The counter will count the number of unknown high frequency pulses during a period of known signal. Unknown Signal → TACLK (Timer0).
- 1 second Reference Period → Timer1



Frequency Measurement Test Setup

LCD1



Selecting Timer Function on P1.0

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ^[1]				
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x=1 ^[2]	CAPD.y
P1.0/ TA0CLK/ ACLK/ A0 ^[2] / CA0/ Pin Osc	0	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.TACLK	0	1	0	0	0
		ACLK	1	1	0	0	0
		A0	X	X	X	1 (y = 0)	0
		CA0	X	X	X	0	1 (y = 0)
		Capacitive sensing	X	0	1	0	0
P1.1/ TA0.0/ UCA0RXD/ UCA0SOMI/ A1 ^[2] / CA1/ Pin Osc	1	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.0	1	1	0	0	0
		TA0.CCI0A	0	1	0	0	0
		UCA0RXD	from USCI	1	1	0	0
		UCA0SOMI	from USCI	1	1	0	0
		A1	X	X	X	1 (y = 1)	0
		CA1	X	X	X	0	1 (y = 1)
		Capacitive sensing	X	0	1	0	0
P1.2/ TA0.1/ UCA0TXD/ UCA0SIMO/ A2 ^[2] / CA2/ Pin Osc	2	P1.x (I/O)	I: 0; O: 1	0	0	0	0
		TA0.1	1	1	0	0	0
		TA0.CCI1A	0	1	0	0	0
		UCA0TXD	from USCI	1	1	0	0
		UCA0SIMO	from USCI	1	1	0	0
		A2	X	X	X	1 (y = 2)	0
		CA2	X	X	X	0	1 (y = 2)
		Capacitive sensing	X	0	1	0	0

```
1 #include <msp430.h>
2 #include <inttypes.h>
3 #include <stdio.h>
4
5 #define CMD      0
6 #define DATA     1
7
8 #define LCD_OUT    P1OUT
9 #define LCD_DIR   P1DIR
10 #define D4        BIT4
11 #define D5        BIT5
12 #define D6        BIT6
13 #define D7        BIT7
14 #define RS       BIT1
15 #define EN       BIT3
16
17 volatile unsigned char count = 0;
18 volatile unsigned int freq; // Global variables
19
20 /**
21 * @brief Delay function for producing delay in 0.1 ms increments
22 * @param t milliseconds to be delayed
23 * @return void
24 */
25 void delay(uint16_t t)
26 {
27     uint16_t i;
28     for(i=t; i > 0; i--)
29         __delay_cycles(100);
30 }
31
32 /**
33 * @brief Function to pulse EN pin after data is written
34 * @return void
35 */
36 void pulseEN(void)
37 {
38     LCD_OUT |= EN;
39     delay(1);
40     LCD_OUT &= ~EN;
41     delay(1);
42 }
43
```

```
43 /**
44 * @brief Function to write data/command to LCD
45 * @param value Value to be written to LED
46 * @param mode Mode -> Command or Data
47 * @return void
48 */
49 void lcd_write(uint8_t value, uint8_t mode)
50 {
51     if(mode == CMD)
52         LCD_OUT &= ~RS;                      // Set RS -> LOW for Command mode
53     else
54         LCD_OUT |= RS;                     // Set RS -> HIGH for Data mode
55
56     LCD_OUT = ((LCD_OUT & 0x0F) | (value & 0xF0));           // Write high nibble first
57     pulseEN();
58     delay(1);
59
60     LCD_OUT = ((LCD_OUT & 0x0F) | ((value << 4) & 0xF0));    // Write low nibble next
61     pulseEN();
62     delay(1);
63 }
64
65 /**
66 * @brief Function to print a string on LCD
67 * @param *s pointer to the character to be written.
68 * @return void
69 */
70 void lcd_print(char *s)
71 {
72     while(*s)
73     {
74         lcd_write(*s, DATA);
75         s++;
76     }
77 }
78
```

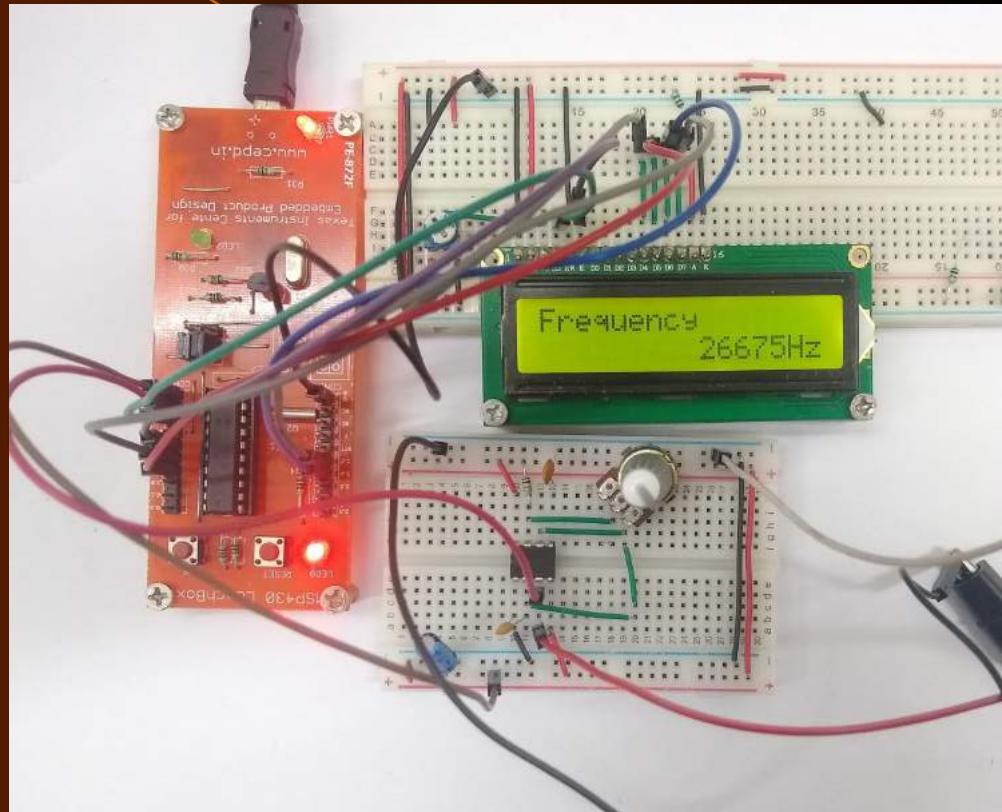
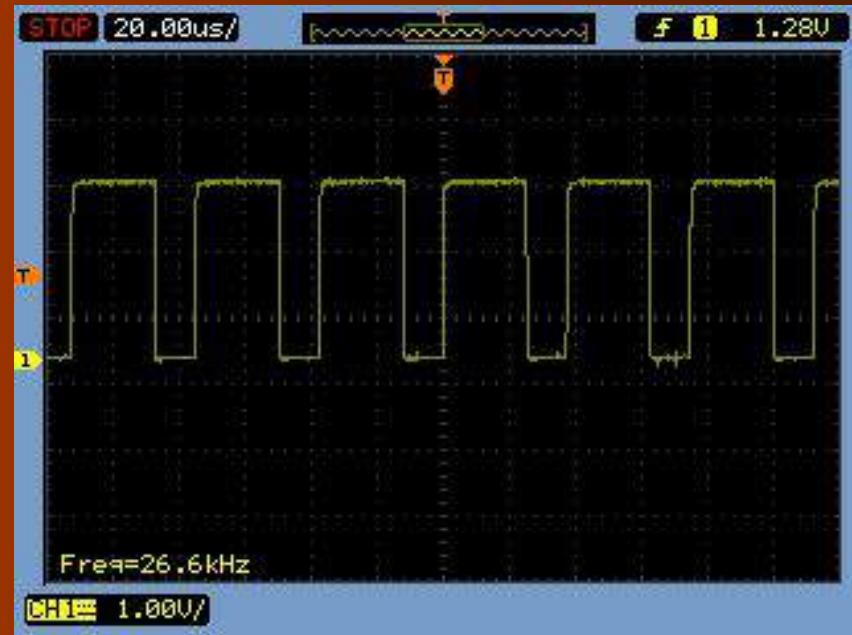
```
78
79 /**
80 * @brief Function to convert number into character array
81 * @param num integer number to be converted.
82 * @return void
83 */
84 void lcd_printNumber(unsigned int num)
85 {
86     char buf[6];
87     char *str = &buf[5];
88
89     *str = '\0';
90
91     do
92     {
93         unsigned long m = num;
94         num /= 10;
95         char c = (m - 10 * num) + '0';
96         *--str = c;
97     } while(num);
98
99     lcd_print(str);
100}
101
102 /**
103 * @brief Function to move cursor to desired position on LCD
104 * @param row Row Cursor of the LCD
105 * @param col Column Cursor of the LCD
106 * @return void
107 */
108 void lcd_setCursor(uint8_t row, uint8_t col)
109 {
110     const uint8_t row_offsets[] = { 0x00, 0x40 };
111     lcd_write(0x80 | (col + row_offsets[row]), CMD);
112     delay(1);
113 }
114
```

```
115 /**
116 * @brief Initialize LCD
117 */
118 void lcd_init()
119 {
120     LCD_DIR |= (D4+D5+D6+D7+RS+EN);
121     LCD_OUT &= ~(D4+D5+D6+D7+RS+EN);
122
123     delay(150);                      // Wait for power up ( 15ms )
124     lcd_write(0x33, CMD);            // Initialization Sequence 1
125     delay(50);                      // Wait ( 4.1 ms )
126     lcd_write(0x32, CMD);            // Initialization Sequence 2
127     delay(1);                       // Wait ( 100 us )
128
129 // All subsequent commands take 40 us to execute, except clear & cursor return (1.64 ms)
130 lcd_write(0x28, CMD);            // 4 bit mode, 2 line
131 delay(1);
132
133 lcd_write(0x0C, CMD);          // Display ON, Cursor OFF, Blink OFF
134 delay(1);
135
136 lcd_write(0x01, CMD);          // Clear screen
137 delay(20);
138
139 lcd_write(0x06, CMD);          // Auto Increment Cursor
140 delay(1);
141 lcd_setCursor(0,0);            // Goto Row 1 Column 1
142 }
143 /**
144 * @brief Displays on LCD
145 */
146 void lcd_display()
147 {
148     lcd_write(0x01, CMD);          // Clear screen
149     delay(20);
150     lcd_setCursor(0,0);
151     lcd_print("Frequency");
152     lcd_setCursor(1,9);
153     lcd_printNumber(freq);
154     lcd_print("Hz");
155     delay(100);
156 }
```

```
158
159 /**
160 * @brief
161 * These settings are w.r.t enabling TIMER1 on Lunch Box
162 */
163 void register_settings_for_TIMER1()
164 {
165     TA1CCTL0 = CCIE;                      // CCR0 interrupt enabled
166     TA1CCR0 = 32768;                     // 1 Hz
167     TA1CTL = TASSEL_1 + MC_1 + TACLR;    // ACLK = 32768 Hz, upmode
168 }
169
170 /*@brief entry point for the code*/
171 void main(void)
172 {
173     WDTCTL = WDTPW + WDTHOLD;           //! Stop Watchdog (Not recommended for code in production and devices working in field)
174
175     unsigned int i;
176
177     do{
178         IFG1 &= ~OFIFG;                  // Clear oscillator fault flag
179         for (i = 50000; i; i--);       // Delay
180     } while (IFG1 & OFIFG);           // Test osc fault flag
181
182     P1DIR &=~ BIT0;                  // Set BIT0 as input
183     P1SEL |= BIT0;                   // Set BIT0 as Timer Clock source
184
185     lcd_init();                      // Initialize LCD
186     register_settings_for_TIMER1();   // Setting for Timer1 as 1 second timer.
187     while(1)
188     {
189         TA1CTL |= TACLR;             // Timer1 value clear
190         __bis_SR_register(LPM0_bits + GIE); // Enter into low power until capturing is done
191         lcd_display();              // Display on 16*2 LCD
192     }
193 }
194 }
```

```
195 /*@brief entry point for TIMER1 interrupt vector*/
196 #pragma vector= TIMER1_A0_VECTOR
197 __interrupt void Timer_A (void)
198 {
199     if(count == 0)
200     {
201         count++;                                // Count = 1
202         TA0CTL |= TASSEL_0 + MC_2 + TACLR;      // Timer0 setting as counter
203                                         // Clock -> TACLK, Cont. Mode, Clear Timer
204     }
205     else
206     {
207         TA0CTL = MC_0;                          // Stop Timer0 count increment
208         freq = TAR;                           // Get counter value
209         count = !count;                      // Count = 0
210         __bic_SR_register_on_exit(LPM0_bits + GIE); // Exit LPM0 on return to main
211     }
212 }
213 }
```

Frequency Measurement Test Setup





Thank you!

Introduction to Embedded System Design

Code like a Ninja

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

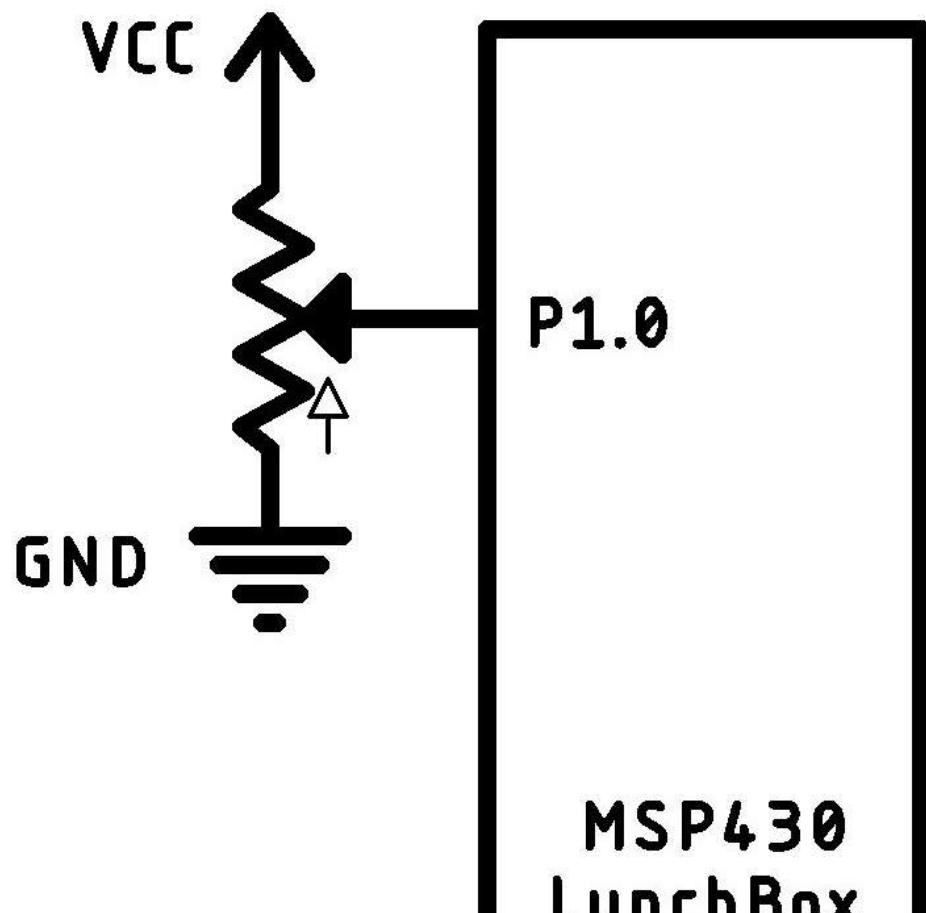
Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Coding Exercise - A1 (Serial Print)

Take Log base 10 of the input analog value and then multiply by two, giving a number from 0 to 6. This value is then printed on Serial Monitor.



```
1#include <msp430.h>
2#include <math.h>
3#include <Library/LunchboxCommon.h>
4#include <stdio.h>
5#include <string.h>
6#include <inttypes.h>
7
8#define AIN      BIT0          // Analog Input at P1.0
9
10volatile float logValues = 0;
11volatile unsigned char number = 1;
12
13/**
14 * @brief
15 * These settings are w.r.t. enabling ADC10 on LunchBox
16 */
17void register_settings_for_ADC10()
18{
19    ADC10AE0 |= AIN;           // P1.0 ADC option select
20    ADC10CTL1 = INCH_0;        // ADC Channel -> 1 (P1.0)
21    ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON; // Ref -> Vcc, 64 CLK S&H
22}
23
```

```
24
25 /*@brief entry point for the code*/
26 void main(void) {
27     WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
28
29
30     BCSCTL1 |= (BIT0 + BIT1 + BIT2 + BIT3); // Selecting RSELx as 15
31     DCOCTL |= (BIT6 + BIT5 + BIT4); // Selecting DCOx as 7, DCO_freq = 15.6 MHz
32
33     register_settings_for_ADC10(); // Register setting for ADC10
34
35     initialise_SerialPrint_on_lunchbox(); // Function to initialize Serial on LunchBox using LunchboxCommon.h
36
37     unsigned int i;
38     while(1)
39     {
40         ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
41         while(ADC10CTL1 & ADC10BUSY); // Wait for conversion to end
42
43         int adcValue = ADC10MEM;
44
45         if(adcValue != 0)
46         {
47             logValues = log10(adcValue); // Taking Log base 10 of ADC value
48             logValues = 2.0 * logValues;
49         }
50
51         number = 1 * logValues;
52         printf("ADC value : %d, \t(Log base 10) * 2 : %c \r\n", adcValue, number + '0');
53
54         for (i = 0; i < 20000; i++);
55     }
56 }
```

COM3 - PuTTY

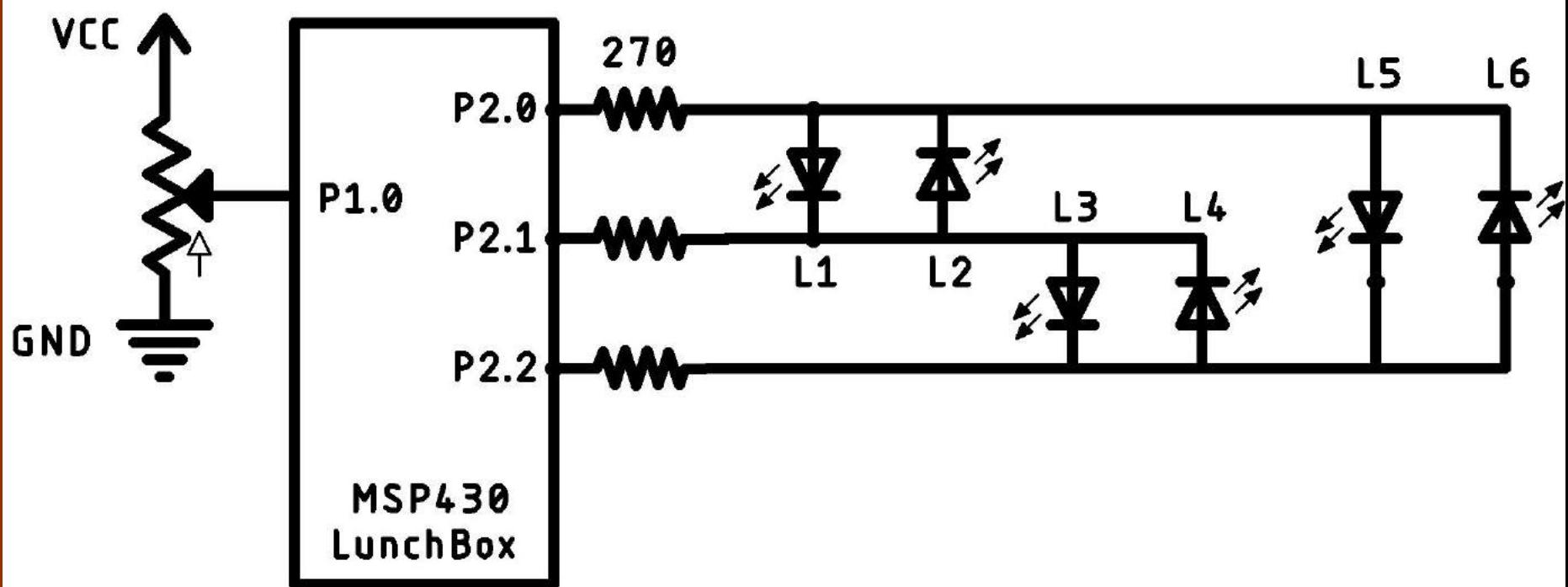
```
ADC value : 848,          (Log base 10) * 2 : 5
ADC value : 847,          (Log base 10) * 2 : 5
ADC value : 845,          (Log base 10) * 2 : 5
ADC value : 851,          (Log base 10) * 2 : 5
ADC value : 848,          (Log base 10) * 2 : 5
ADC value : 851,          (Log base 10) * 2 : 5
ADC value : 845,          (Log base 10) * 2 : 5
ADC value : 851,          (Log base 10) * 2 : 5
ADC value : 843,          (Log base 10) * 2 : 5
ADC value : 851,          (Log base 10) * 2 : 5
ADC value : 850,          (Log base 10) * 2 : 5
ADC value : 845,          (Log base 10) * 2 : 5
ADC value : 851,          (Log base 10) * 2 : 5
ADC value : 850,          (Log base 10) * 2 : 5
ADC value : 843,          (Log base 10) * 2 : 5
ADC value : 849,          (Log base 10) * 2 : 5
ADC value : 846,          (Log base 10) * 2 : 5
ADC value : 848,          (Log base 10) * 2 : 5
ADC value : 848,          (Log base 10) * 2 : 5
ADC value : 850,          (Log base 10) * 2 : 5
ADC value : 851,          (Log base 10) * 2 : 5
ADC value : 846,          (Log base 10) * 2 : 5
ADC value : 851,          (Log base 10) * 2 : 5
```

Coding Exercise - A2

(Charlieplexed LEDs)

Implement a Charlieplexed display which turn on LEDs from 0 to 6 on potentiometer rotation. Log base 10 of the input value Analog value is taken and then multiplied by two, giving a number from 0 to 6.

Circuit Diagram/Picture



Charlieplexed LEDs

```
1 #include <msp430.h>
2 #include <math.h>
3
4 #define AIN      BIT0          // Analog Input at P1.0
5
6 #define P1       BIT0          // Charlieplex P1 -> P2.0
7 #define P2       BIT1          // Charlieplex P2 -> P2.1
8 #define P3       BIT2          // Charlieplex P3 -> P2.2
9
10 volatile float logValues = 0;
11 volatile unsigned char number = 1;
12
13 //Data Table for 12 Charlieplexed LEDs
14 const unsigned int hi[6] = {P2,P1,P3,P1,P3,P2};
15 const unsigned int lo[6] = {P1,P2,P1,P3,P2,P3};
16 const unsigned int z[6]  = {P3,P3,P2,P2,P1,P1};
17
18 /**
19 * @brief
20 * These settings are w.r.t. Charlieplexed LEDs
21 */
22 void charlie(unsigned int value)
23 {
24     P2DIR &= ~(z[value]);           // Set high-Z pins as Input
25     P2DIR |= (lo[value] + hi[value]); // Set high & low pins as Output
26     P2OUT &= ~lo[value];          // Set state of low pin
27     P2OUT |= hi[value];          // Set state of high pin
28 }
29
```

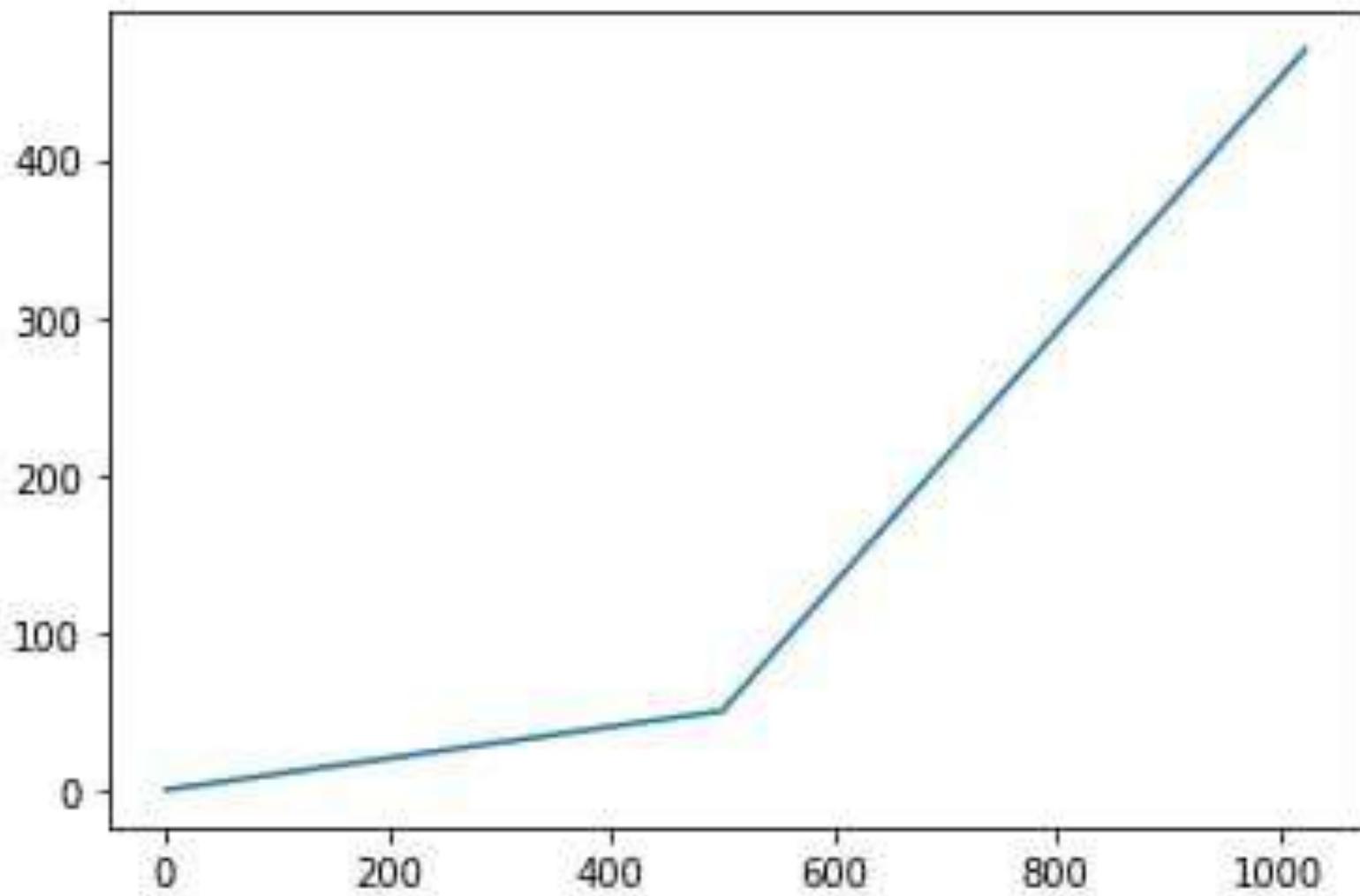
```
29
30 /**
31 * @brief
32 * These settings are w.r.t. enabling ADC10 on LunchBox
33 */
34 void register_settings_for_ADC10()
35 {
36     ADC10AE0 |= AIN;                                // P1.0 ADC option select
37     ADC10CTL1 = INCH_0;                            // ADC Channel -> 1 (P1.0)
38     ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON;    // Ref -> Vcc, 64 CLK S&H
39 }
40
41 /**
42 * @brief
43 * These settings are w.r.t. enabling TIMER0 on Lunch Box
44 */
45 void register_settings_for_TIMER0()
46 {
47     CCTL0 = CCIE;                                 // CCR0 interrupt enabled
48     TACTL = TASSEL_1 + MC_1;                      // ACLK = 32768 Hz, up mode
49     CCR0 = 32;                                    // 1024 Hz
50 }
51
```

```
52 /*@brief entry point for the code*/
53 void main(void) {
54     WDTCTL = WDTPW | WDTHOLD;                                // Stop watchdog timer
55
56
57     BCSCTL1 |= (BIT0 + BIT1 + BIT2 + BIT3);                // Selecting RSELx as 15
58     DCOCTL  |= (BIT6 + BIT5 + BIT4);                      // Selecting DCOx as 7, DCO_freq = 15.6 MHz
59
60     register_settings_for_ADC10();
61     register_settings_for_TIMER0();
62
63     __bis_SR_register(GIE);                                // Enable CPU Interrupt
64
65     while(1)
66     {
67         ADC10CTL0 |= ENC + ADC10SC;                        // Sampling and conversion start
68         while(ADC10CTL1 & ADC10BUSY);                     // Wait for conversion to end
69
70         int adcValue = ADC10MEM;
71         if(adcValue != 0)
72         {
73             logValues = log10(adcValue);                  // Taking Log base 10 of ADC value
74             logValues = 2.0 * logValues;
75         }
76     }
77 }
78
79 /*@brief entry point for TIMER0 interrupt vector*/
80 #pragma vector= TIMER0_A0_VECTOR
81 __interrupt void Timer_A (void)
82 {
83     if(number <= logValues)
84     {
85         charlie(number-1);                                //Switch on LED (i)
86         number++;
87     }
88     else
89     {
90         number = 1;
91     }
92 }
```

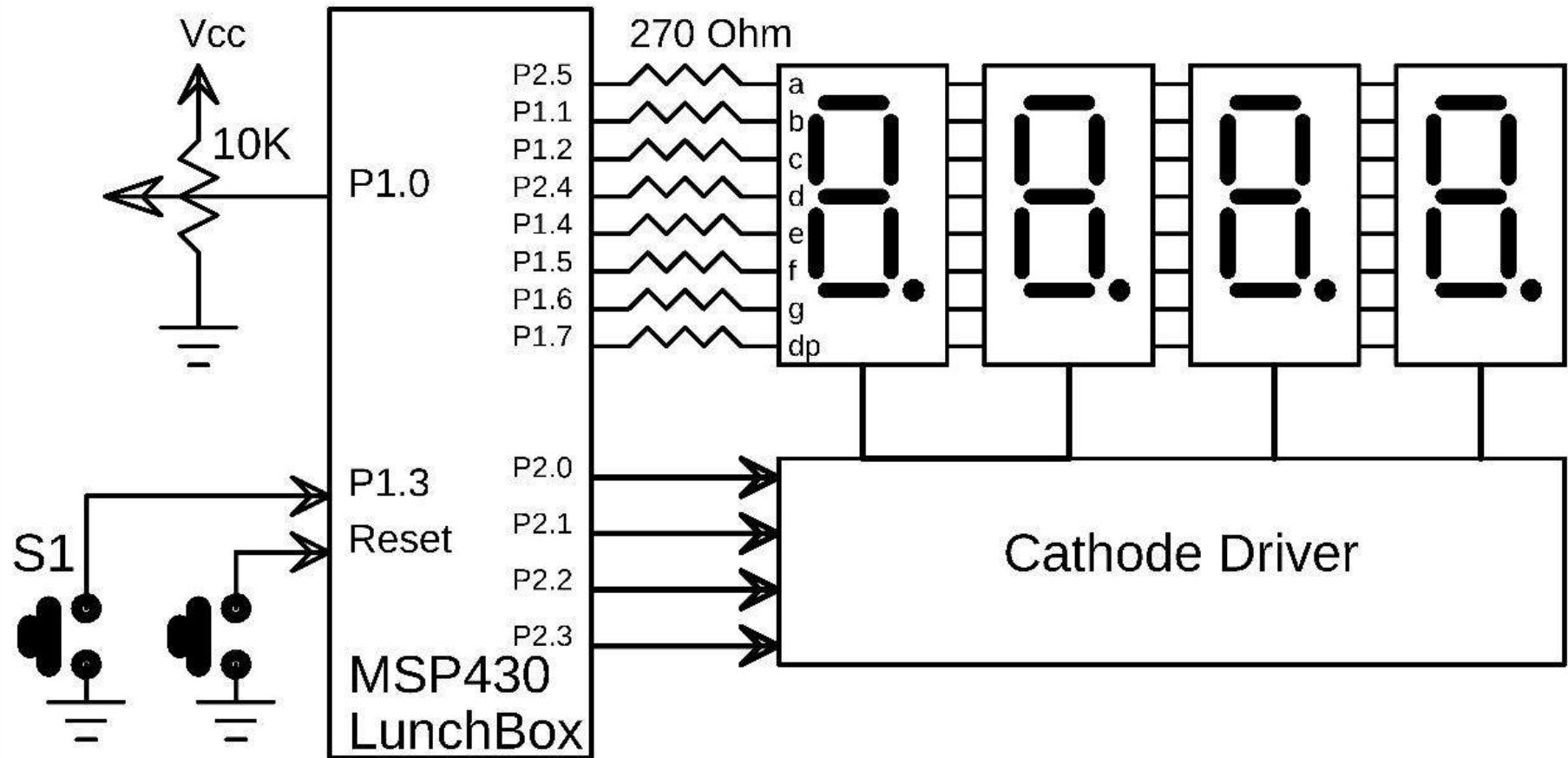
Advanced Coding

Exercise - B

Implement a Hexadecimal Counter which counts from 0000 to 9999 on switch press. Display the count on the 4-digit SSD. Change the refresh rate of the 4-digit SSD with the help of a Potentiometer input.



Circuit Diagram/Picture



Coding style 1: Sequential execution

```
1 #include <msp430.h>
2 #include <inttypes.h>
3
4 #define AIN      BIT0          // Potentiometer -> P1.0
5
6 #define SW       BIT3          // Switch -> P1.3
7
8 // Define Pin Mapping of 7-segment Display
9 // Segment A is connected to P2.5, Segment D is connected to P2.4
10 // Segments B,C,E,F,G and DP are connected to P1.1, P1.2, P1.4, P1.5, P1.6, P1.7 respectively
11 #define SEG_A    BIT5
12 #define SEG_B    BIT1
13 #define SEG_C    BIT2
14 #define SEG_D    BIT4
15 #define SEG_E    BIT4
16 #define SEG_F    BIT5
17 #define SEG_G    BIT6
18 #define SEG_DP   BIT7
19
20 // Segments 1-4 are connected to P2.0 - P2.3
21 #define SEG_1    BIT0
22 #define SEG_2    BIT1
23 #define SEG_3    BIT2
24 #define SEG_4    BIT3
25
26 volatile unsigned int displayValue = 0;
27 volatile float delayValue = 0;
28
29 /**
30 * @brief Delay function for producing delay in .1 ms increments
31 * @param t Input time to be delayed
32 * @return void
33 */
34 void delay(uint16_t t)
35 {
36     uint16_t i;
37     for(i=t; i > 0; i--)
38         delay_cycles(1500);
39 }
40
```

```
45 */
46 void digitToDisplay(unsigned int numberToBeDisplayed){
47
48     switch(numberToBeDisplayed){
49         case 0: P2OUT |= (SEG_A + SEG_D);
50             P1OUT |= SEG_B + SEG_C + SEG_E + SEG_F;
51             break;
52
53         case 1: P1OUT |= SEG_B + SEG_C;
54             break;
55
56         case 2: P2OUT |= (SEG_A + SEG_D);
57             P1OUT |= SEG_B + SEG_E + SEG_G;
58             break;
59
60         case 3: P2OUT |= (SEG_A + SEG_D);
61             P1OUT |= SEG_B + SEG_C + SEG_G;
62             break;
63
64         case 4: P1OUT |= SEG_B + SEG_C + SEG_F + SEG_G;
65             break;
66
67         case 5: P2OUT |= (SEG_A + SEG_D);|
68             P1OUT |= SEG_C + SEG_F + SEG_G;
69             break;
70
71         case 6: P2OUT |= (SEG_A + SEG_D);
72             P1OUT |= SEG_C + SEG_E + SEG_F + SEG_G;
73             break;
74
75         case 7: P2OUT |= SEG_A;
76             P1OUT |= SEG_B + SEG_C;
77             break;
78
79         case 8: P2OUT |= (SEG_A + SEG_D);
80             P1OUT |= SEG_B + SEG_C + SEG_E + SEG_F + SEG_G;
81             break;
82
83         case 9: P2OUT |= (SEG_A + SEG_D);
84             P1OUT |= SEG_B + SEG_C + SEG_F + SEG_G;
85             break;
86     }
87 }
```

```
89 /**
90 * @brief This function separates 4 digits from a number and displayed it on a four digit seven segment display
91 * @param number Number to be displayed
92 * @return Void
93 */
94 void fourDigitNumber(int number)
95 {
96     unsigned int displayDigit[4];
97     displayDigit[0] = number / 1000;
98     number = number % 1000;
99     displayDigit[1] = number / 100;
100    number = number % 100;
101    displayDigit[2] = number / 10;
102    displayDigit[3] = number % 10;
103
104
105    P1OUT &=~ (SEG_B + SEG_C + SEG_E + SEG_F + SEG_G);
106    P2OUT &=~ (SEG_A + SEG_D);
107    P2OUT &=~ (SEG_1 + SEG_2 + SEG_3 + SEG_4);
108
109    digitToDisplay(displayDigit[0]);      // Display first digit
110    P2OUT |= SEG_4;
111    P1OUT &=~ SEG_DP;
112    delay(delayValue);
113    P1OUT ^= SEG_DP;
114
115    P1OUT &=~ (SEG_B + SEG_C + SEG_E + SEG_F + SEG_G);
116    P2OUT &=~ (SEG_A + SEG_D);
117    P2OUT &=~ (SEG_1 + SEG_2 + SEG_3 + SEG_4);
118
119    digitToDisplay(displayDigit[1]);      // Display second digit
120    P2OUT |= SEG_3;
121    delay(delayValue);
122
123    P1OUT &=~ (SEG_B + SEG_C + SEG_E + SEG_F + SEG_G);
124    P2OUT &=~ (SEG_A + SEG_D);
125    P2OUT &=~ (SEG_1 + SEG_2 + SEG_3 + SEG_4);
```

```
124
125     digitToDisplay(displayDigit[2]);      // Display third digit
126     P2OUT |= SEG_2;
127     delay(delayValue);
128
129     P1OUT &=~ (SEG_B + SEG_C + SEG_E + SEG_F + SEG_G);
130     P2OUT &=~ (SEG_A + SEG_D);
131     P2OUT &=~ (SEG_1 + SEG_2 + SEG_3 + SEG_4);
132
133     digitToDisplay(displayDigit[3]);      // Display forth digit
134     P2OUT |= SEG_1;
135     delay(delayValue);
136 }
137
138 /**
139 * @brief
140 * These settings are wrt enabling ADC10 on Lunchbox
141 */
142 void register_settings_for_ADC10()
143 {
144     ADC10AE0 |= AIN;                      // P1.0 ADC option select
145     ADC10CTL1 = INCH_0;                   // ADC Channel -> 1 (P1.0)
146     ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON; // Ref -> Vcc, 64 CLK S&H
147 }
148
```

```
162 /**
163 * @brief
164 * These settings are w.r.t GPIO on Lunch Box
165 */
166 void initialize_GPIO()
167 {
168     // Setting Pins connected to SSD as OUTPUT
169     P1DIR |= (SEG_B + SEG_C + SEG_E + SEG_F + SEG_G + SEG_DP);
170     P2DIR |= (SEG_A + SEG_D);
171     P2DIR |= (SEG_1 + SEG_2 + SEG_3 + SEG_4);
172
173     // Setting Pins connected to SSD as LOW
174     P1OUT &=~ (SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G);
175     P2OUT &=~ (SEG_A + SEG_D);
176     P2OUT &=~ (SEG_1 + SEG_2 + SEG_3 + SEG_4);
177
178     // Setting Switch pin as input
179     P1DIR &=~ (SW);
180 }
181
182
183 /*@brief entry point for the code*/
184 void main(void) {
185     WDTCTL = WDTPW | WDTHOLD;           //! Stop Watchdog (Not recommended for code in production and devices working in field)
186
187     BCSCTL1 |= (BIT0 + BIT1 + BIT2 + BIT3); // Selecting RSELx as 15
188     DCOCTL  |= (BIT6 + BIT5 + BIT4);      // Selecting DCOx as 7, DCO_freq = 15.6 MHz
189
190     initialize_GPIO();                  // GPIO setting
191
192     register_settings_for_ADC10();      // Setting for ADC10
193 }
```

```
193
194     while(1)
195     {
196         ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
197         while(ADC10CTL1 & ADC10BUSY);        // Wait for conversion to end
198         int adcValue1 = ADC10MEM;            // ADC value 1
199
200         ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
201         while(ADC10CTL1 & ADC10BUSY);        // Wait for conversion to end
202         int adcValue2 = ADC10MEM;            // ADC value 2
203
204         int avgADC = (adcValue1 + adcValue2) / 2; // Average of two values
205
206         if (avgADC < 500)                  // Calculating delay value
207             delayValue = (1 + .1*avgADC) * 10.0;
208         else
209             delayValue = (51 + 0.8*(avgADC - 500)) * 10.0;
210
211
212         if(!(P1IN & SW))                  // If SW is Pressed
213         {
214             delay_cycles(200);           // Wait 20ms to debounce
215             while(!(P1IN & SW));       // Wait till SW Released
216             delay_cycles(200);           // Wait 20ms to debounce
217             displayValue = displayValue + 1;
218             if(displayValue > 9999)
219                 displayValue = 0;
220         }
221
222         fourDigitNumber(displayValue);
223         P2OUT &=~ (SEG_1 + SEG_2 + SEG_3 + SEG_4);
224     }
225 }
```

Coding style 2: ISR Based Implementation

```
36 void digitToDisplay(unsigned int numberToBeDisplayed){  
37  
38     switch(numberToBeDisplayed){  
39         case 0: P2OUT |= (SEG_A + SEG_D);  
40             P1OUT |= SEG_B + SEG_C + SEG_E + SEG_F;  
41             break;  
42  
43         case 1: P1OUT |= SEG_B + SEG_C;  
44             break;  
45  
46         case 2: P2OUT |= (SEG_A + SEG_D);  
47             P1OUT |= SEG_B + SEG_E + SEG_G;  
48             break;  
49  
50         case 3: P2OUT |= (SEG_A + SEG_D);  
51             P1OUT |= SEG_B + SEG_C + SEG_G;  
52             break;  
53  
54         case 4: P1OUT |= SEG_B + SEG_C + SEG_F + SEG_G;  
55             break;  
56  
57         case 5: P2OUT |= (SEG_A + SEG_D);  
58             P1OUT |= SEG_C + SEG_F + SEG_G;  
59             break;  
60  
61         case 6: P2OUT |= (SEG_A + SEG_D);  
62             P1OUT |= SEG_C + SEG_E + SEG_F + SEG_G;  
63             break;  
64  
65         case 7: P2OUT |= SEG_A;  
66             P1OUT |= SEG_B + SEG_C;  
67             break;  
68  
69         case 8: P2OUT |= (SEG_A + SEG_D);  
70             P1OUT |= SEG_B + SEG_C + SEG_E + SEG_F + SEG_G;  
71             break;  
72  
73         case 9: P2OUT |= (SEG_A + SEG_D);  
74             P1OUT |= SEG_B + SEG_C + SEG_F + SEG_G;  
75             break;  
76     }  
77 }
```

```
79 /**
80 * @brief This function separates 4 digits from a number.
81 * @param number Number to be displayed
82 * @return Void
83 */
84 void fourDigitNumber(int number)
85 {
86     displayDigit[0] = number / 1000;
87     number = number % 1000;
88     displayDigit[1] = number / 100;
89     number = number % 100;
90     displayDigit[2] = number / 10;
91     displayDigit[3] = number % 10;
92 }
93
94 /**
95 * @brief This function displays the number on a four digit seven segment display
96 * @param number Number to be displayed
97 * @return Void
98 */
99 void displayNumber()
100 {
101     P1OUT &=~ (SEG_B + SEG_C + SEG_E + SEG_F + SEG_G);
102     P2OUT &=~ (SEG_A + SEG_D);
103     P2OUT &=~ (SEG_1 + SEG_2 + SEG_3 + SEG_4);
104
105     switch(digit)
106     {
107         case 4:
108             digitToDisplay(displayDigit[0]);      // Display first digit
109             P2OUT |= SEG_4;
110             break;
111         case 3:
112             digitToDisplay(displayDigit[1]);      // Display second digit
113             P2OUT |= SEG_3;
114             break;
115         case 2:
116             digitToDisplay(displayDigit[2]);      // Display third digit
117             P2OUT |= SEG_2;
118             break;
```

```
119     case 1:  
120         digitToDisplay(displayDigit[3]);      // Display forth digit  
121         P2OUT |= SEG_1;  
122         break;  
123     }  
124 }  
125  
126 /**  
127 * @brief  
128 * These settings are wrt enabling ADC10 on Lunchbox  
129 **/  
130 void register_settings_for_ADC10()  
131 {  
132     ADC10AE0 |= AIN;                      // P1.0 ADC option select  
133     ADC10CTL1 = INCH_0;                   // ADC Channel -> 1 (P1.0)  
134     ADC10CTL0 = ADC10IE;                  // Set ADC module  
135     ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON; // Ref -> Vcc, 64 CLK S&H, ADC10 enable  
136 }  
137  
138 /**  
139 * @brief  
140 * These settings are w.r.t enabling TIMER0 on Lunch Box  
141 **/  
142 void register_settings_for_TIMER0()  
143 {  
144     CCTL0 = CCIE;                        // CCR0 interrupt enabled  
145     TACTL = TASSEL_1 + MC_1;             // ACLK = 32768/4 Hz, up to CCR0  
146 }
```

```
161 /**
162 * @brief
163 * These settings are w.r.t GPIO on Lunch Box
164 */
165 void initialize_GPIO()
166 {
167     // Setting Pins connected to SSD as OUTPUT
168     P1DIR |= (SEG_B + SEG_C + SEG_E + SEG_F + SEG_G + SEG_DP);
169     P2DIR |= (SEG_A + SEG_D);
170     P2DIR |= (SEG_1 + SEG_2 + SEG_3 + SEG_4);
171
172     // Setting Pins connected to SSD as LOW
173     P1OUT &= ~(SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G);
174     P2OUT &= ~(SEG_A + SEG_D);
175     P2OUT &= ~(SEG_1 + SEG_2 + SEG_3 + SEG_4);
176
177     // Setting Switch pin as input
178     P1DIR &= ~ (SW);
179 }
180
181 /*@brief entry point for the code*/
182 void main(void) {
183     WDTCTL = WDTPW | WDTHOLD;           //! Stop Watchdog (Not recommended for code in production and devices working in field)
184
185     BCSCTL1 = DIVA_2;                  // ACLK divider : /4
186
187     initialize_GPIO();                // GPIO setting
188
189     register_settings_for_ADC10();    // Setting for ADC10
190
191     register_settings_for_TIMER0();   // Setting for TIMER0
192
193     __bis_SR_register(GIE);          // Enable CPU Interrupt
```

```
194     while(1)
195     {
196         if(!(P1IN & SW)) // If SW is Pressed
197         {
198             delay_cycles(200); // Wait 20ms to debounce
199             while(!(P1IN & SW)); // Wait till SW Released
200             delay_cycles(200); // Wait 20ms to debounce
201             displayValue = displayValue + 1;
202             if(displayValue > 9999)
203                 displayValue = 0;
204         }
205     }
206     ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
207
208     while(ADC10CTL1 & ADC10BUSY); // Wait for conversion to end
209
210     int adcValue = ADC10MEM; // Saving ADC value to adcValue
211
212     if (adcValue <500) // Calculating TIMER0 compare register value
213         CCR0 = (1 + .1 * adcValue) * 10.0;
214     else
215         CCR0 = (51 + 0.8 * (adcValue - 500)) * 10.0;
216
217     fourDigitNumber(displayValue); // Diving number to be displayed into 4 separate digits.
218 }
219 */
220 /*@brief entry point for TIMER0 interrupt vector*/
221 #pragma vector= TIMER0_A0_VECTOR
222 __interrupt void Timer_A (void)
223 {
224     digit--;
225     if(digit == 0)
226         digit = 4;
227     displayNumber(); // Display number according to the digit value
228 }
229 }
```



Thank you!

Introduction to Embedded System Design

Planning and Building an Electronics Project

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

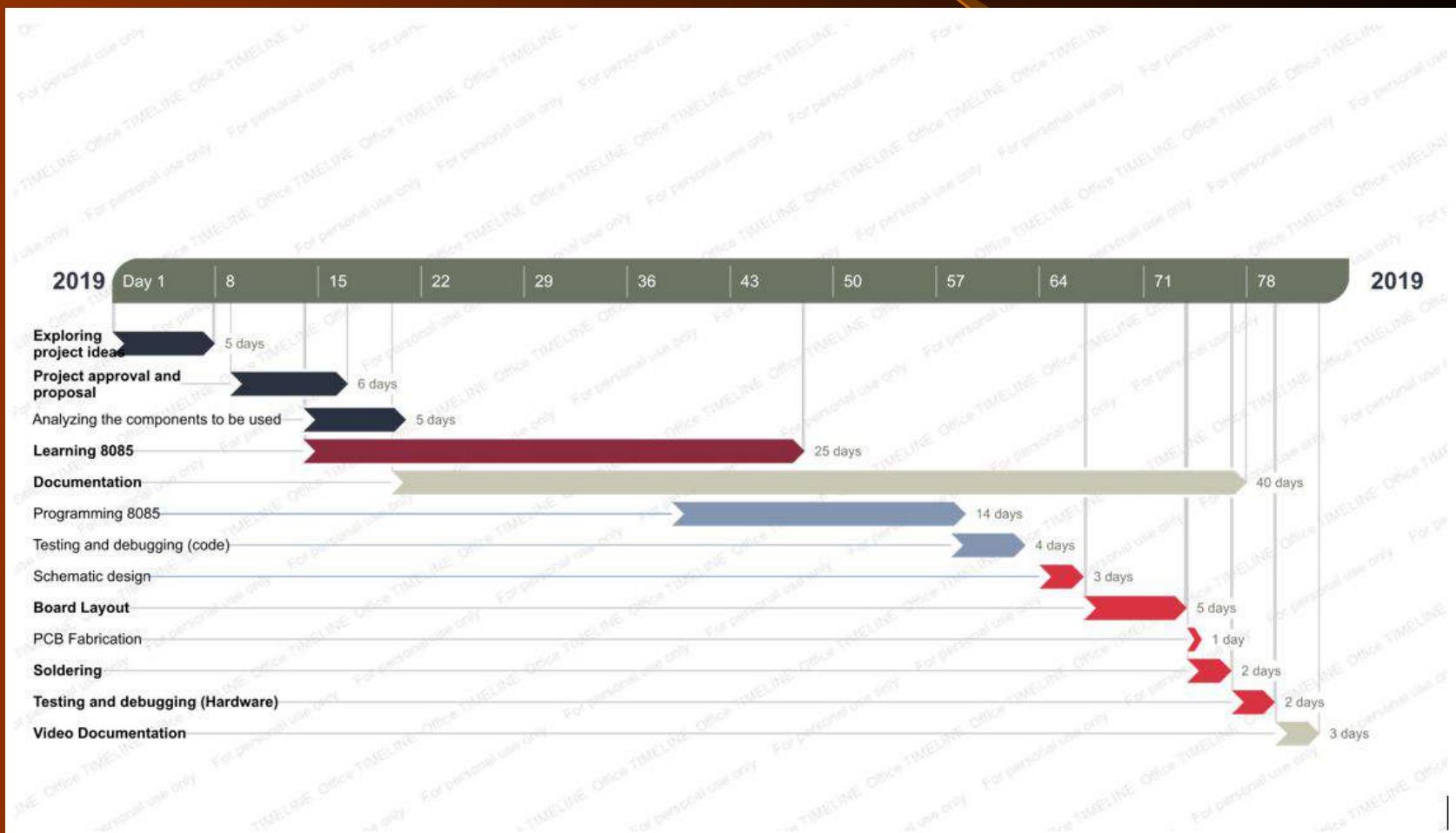
Building an Electronics Project

1. Aim, Objective and Deliverables
2. Visualization
3. Schematic and PCB Layout
4. Circuit Fabrication
5. Power Supply
6. Circuit Soldering
7. System Wiring
8. Testing
9. Enclosures
10. Documentation

Aim, Objective and Deliverables

- What is the aim of project?
- Deliverables
 - Final project in physical form
 - Any prototypes if exist
 - Record of testing strategy and test data
 - Comprehensive report with suitable photographs
- Gantt Chart

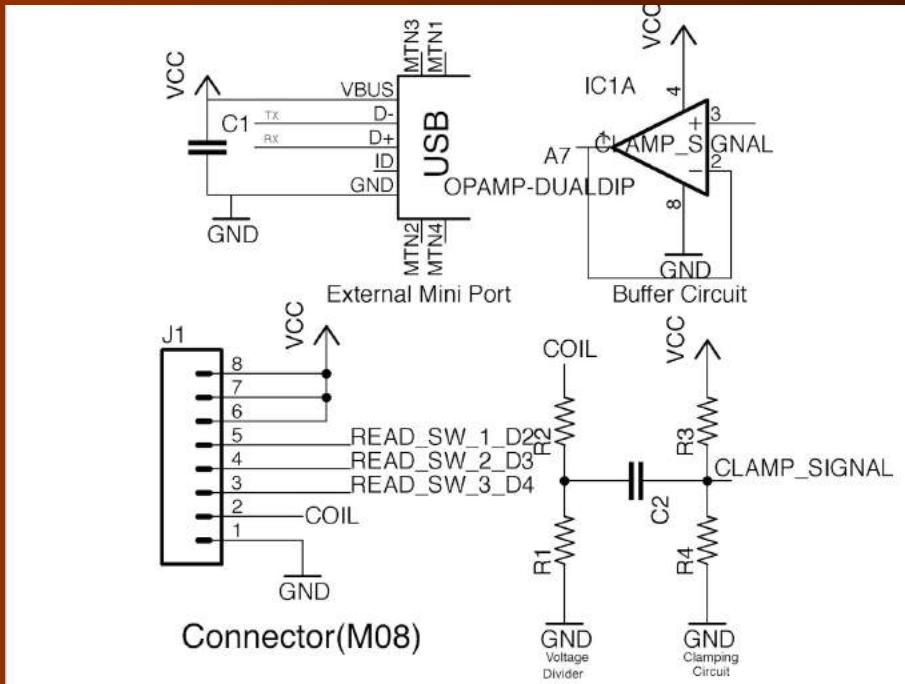
Gantt Chart



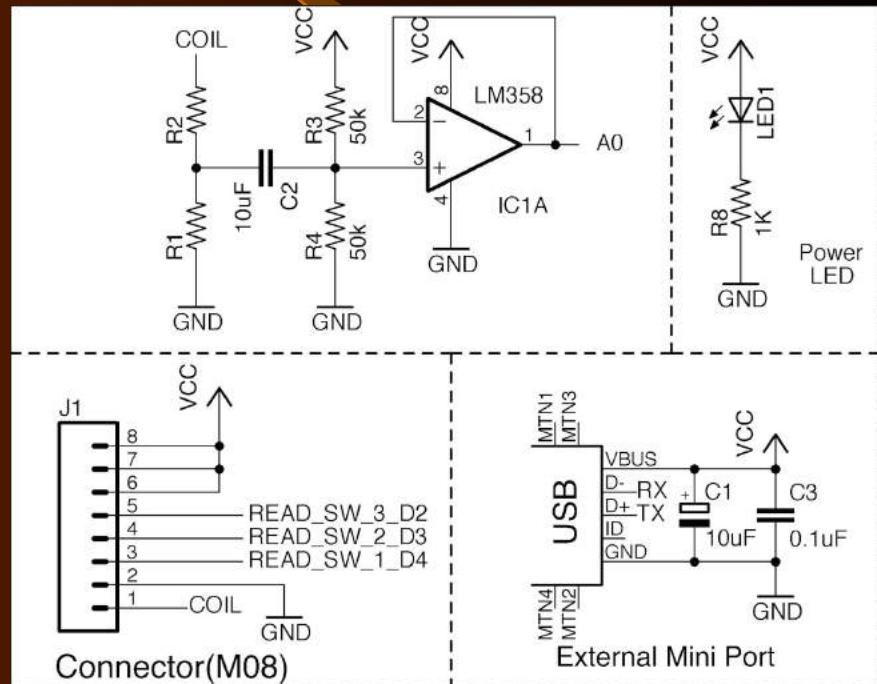
Visualization

- How do you visualize your project.
- Drawing Sketches
- CAD tool

Schematic



Badly Drawn Schematic

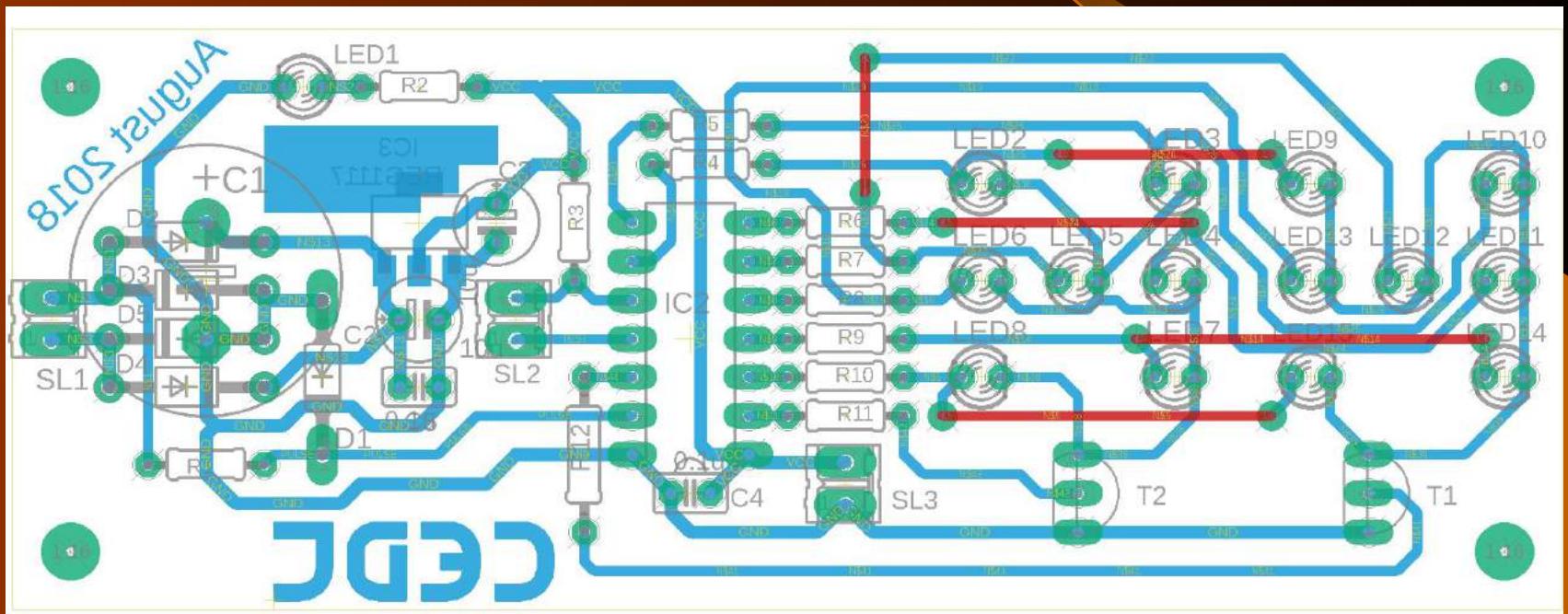


Better Schematic

Bill of Materials

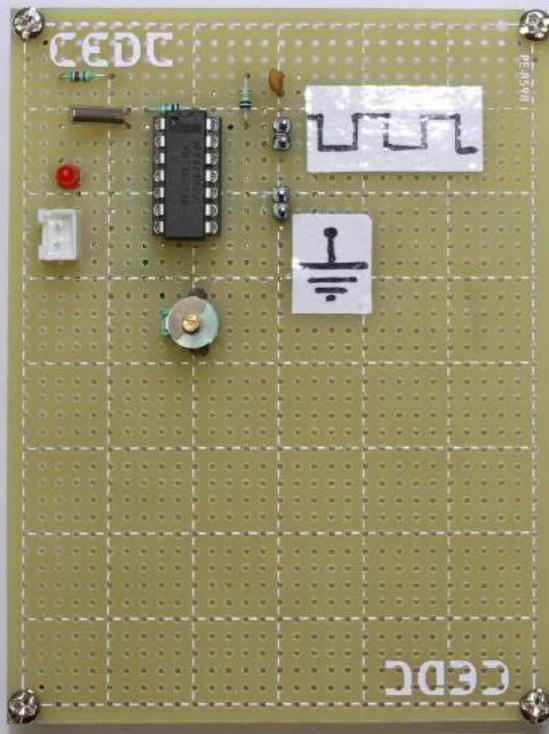
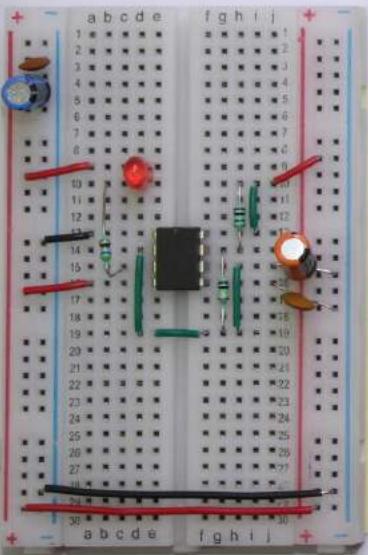
	A	B	C	D
1	Qty	Value	Device	Package
2		1	M03	03P
3	3	0.1u	C-EU025-024X044	C025-024X044
4	1	100u	CPOL-USE2.5-5	E2
5	2	10k	R-US_0204/7	0204/7
6	1	10u	CPOL-USE2.5-5	E2
7	1	15k	R-US_0204/7	0204/7
8	1	1k	R-US_0204/7	0204/7
9	1	20k	R-US_0204/7	0204/7
10	2		330 R-US_0204/7	0204/7
11	1	LM358N	LM358N	DIL08
12	1	LOGO-CEDTBOT-18MM	LOGO-CEDTBOT-18MM	LOGO-CEDT-BOT-18MM
13	1	LOGO-CEDTSILK-18MM	LOGO-CEDTSILK-18MM	LOGO-CEDT-SILK-18MM
14				

PCB Layout



Circuit Prototyping

- BreadBoard
 - Issues?
- General purpose zeroboard
 - SMD components?
- Prototype using PCB made in lab
- PCB fabricated through PCB vendor



Power Supply

- General Purpose Power Supply for testing
- Measuring Voltage and Current
- Digital Multimeter
- Own Power Supply for project

Circuit Soldering

- Inspection of possible shorts or open connections.
- Order of soldering - SMD -> Through hole
- Usage of flux

System Wiring

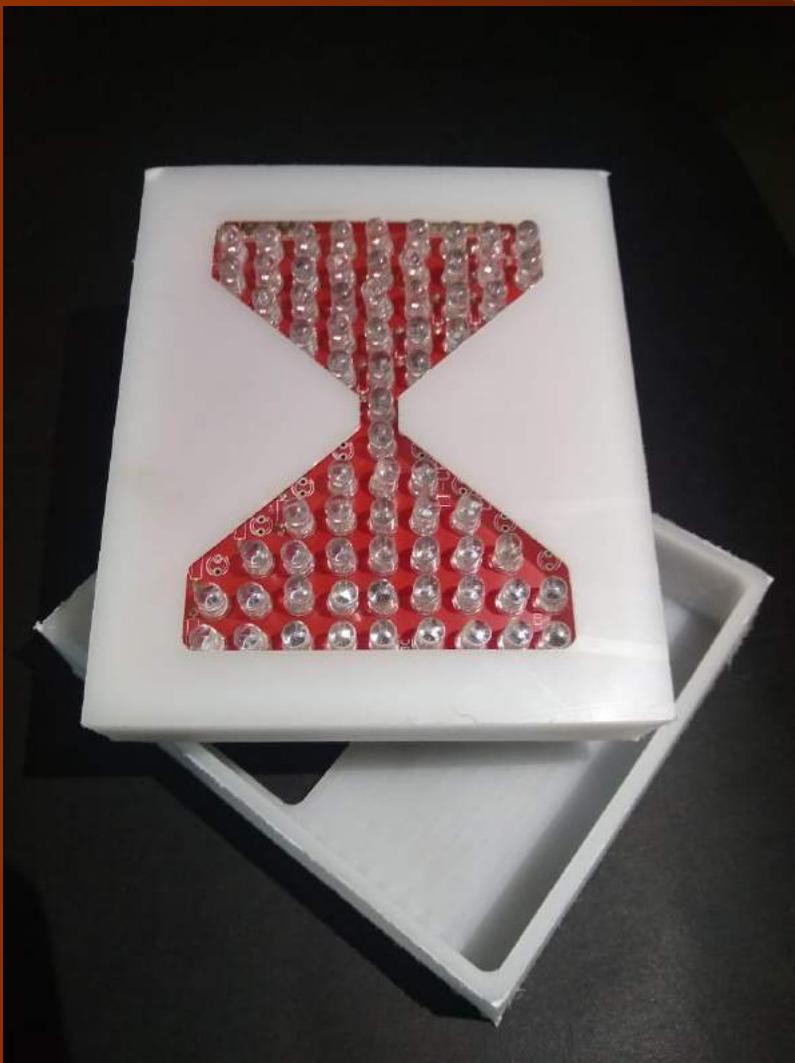
- Connection to outside world.
- Anchoring wires to physical enclosure or PCB

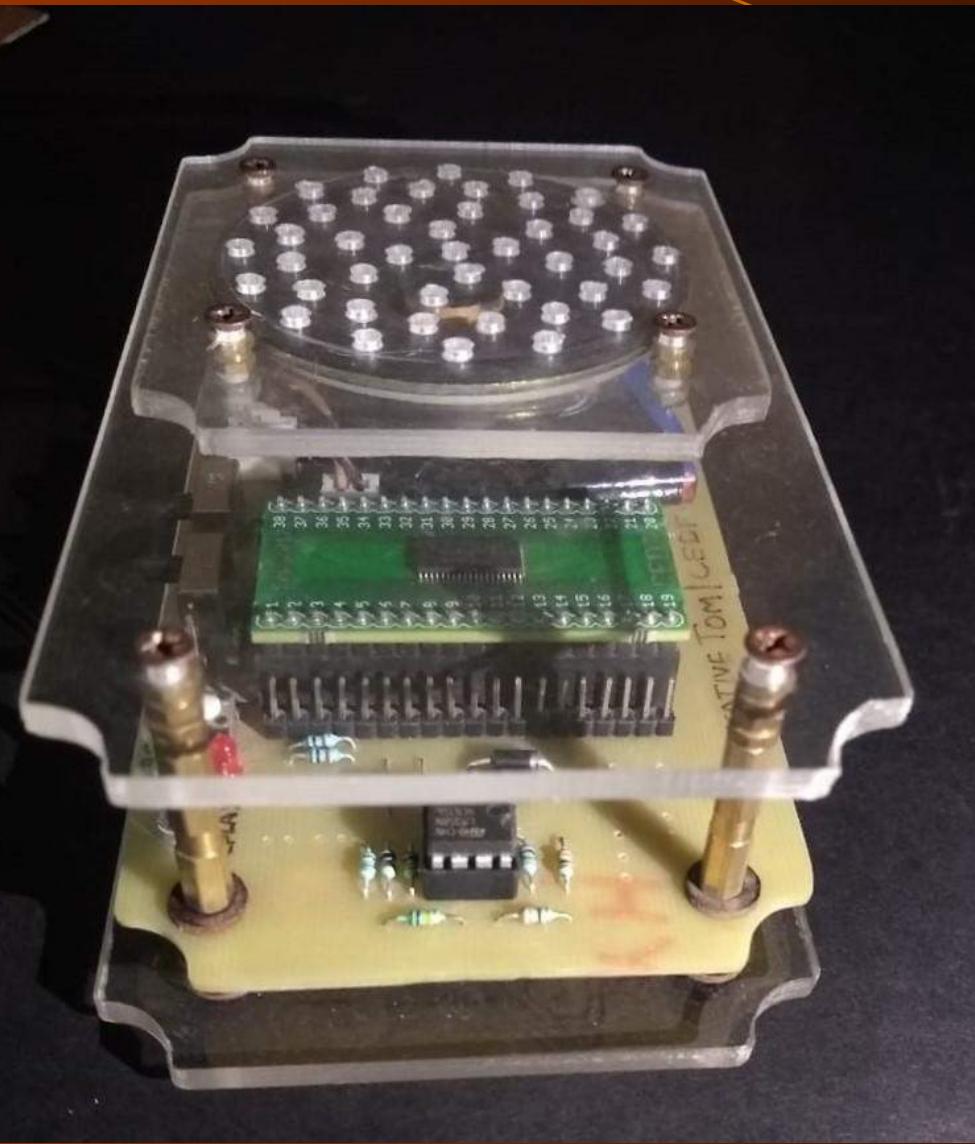
Testing

- Start by testing power Supply
- SMD vs DIP IC testing
- Supply voltage at important points
- Operating Point
- Timed Circuits
- Error voltage at input in op-amps
- Frequency of PWM signal

Enclosures

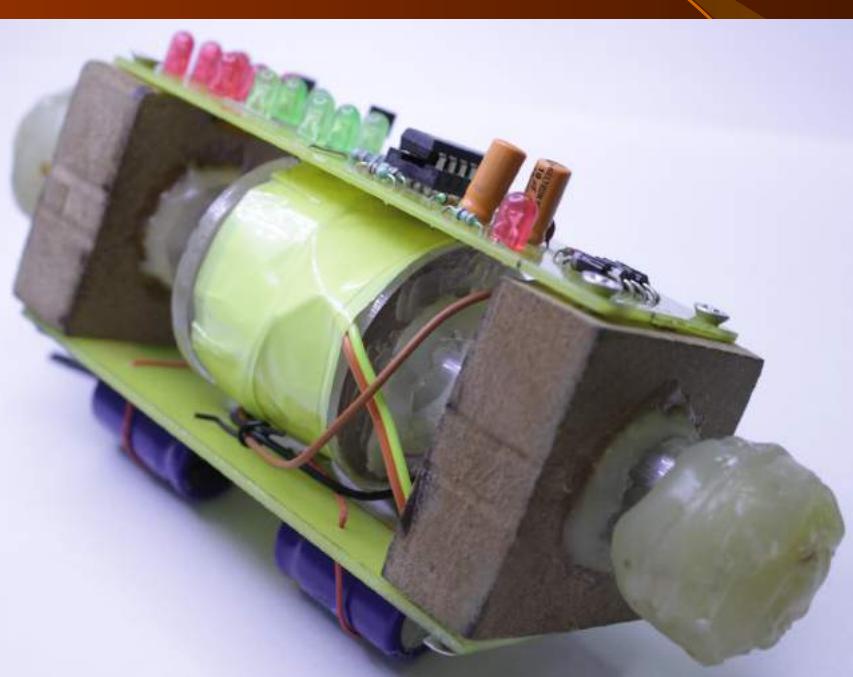
- Ready-Made Enclosure
- Using PCB stock
- 3D printing Custom Enclosure
- Using Paper or Cardboard
- Using Traditional materials (like wood, plastic)
- Using Food Grade Boxes
- Identifier Marks





Documentation

- Title
- Motivation
- Technical details (Block Diagram, electronic circuitry, PCB Layout etc)
- Flowchart, Code
- Pictures
- References
- Video of operation of project.





Thank you!

Introduction to Embedded System Design

Circuit Prototyping Techniques

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

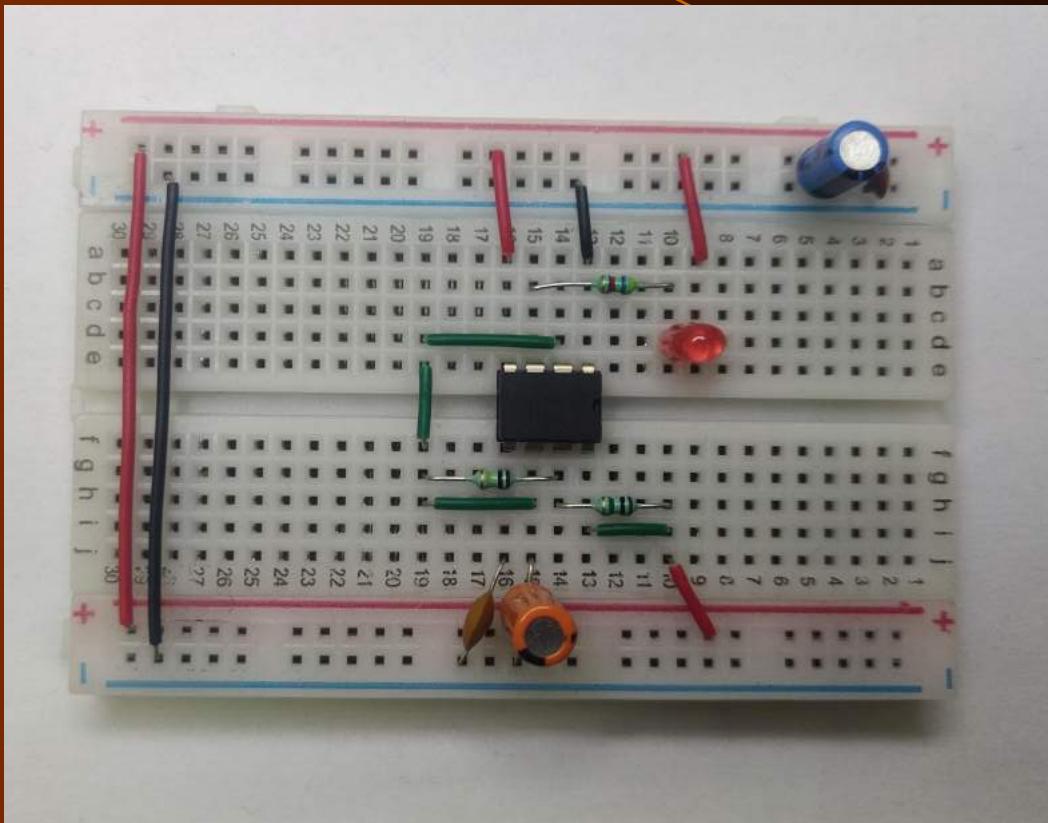
Electrical Engineering Department

Indian Institute of Technology,
Jammu

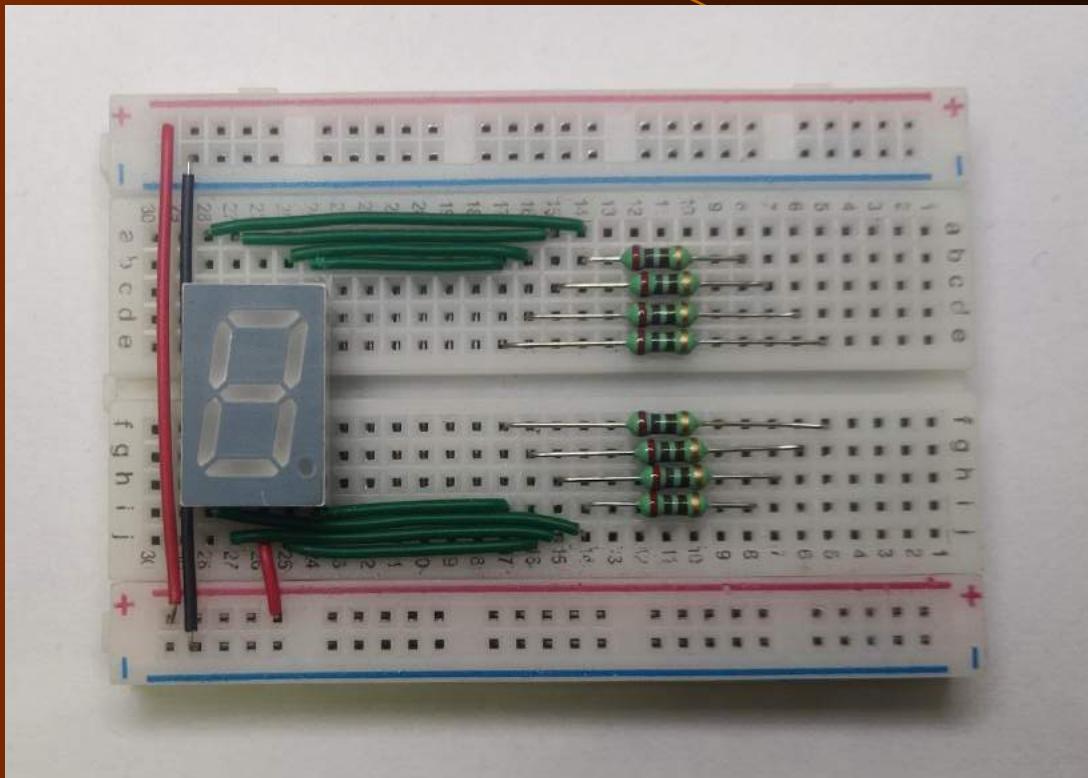
Circuit Prototyping Methods

1. Breadboard
2. Zeroboard
3. Manhattan Style
4. Custom Printed Circuit Board

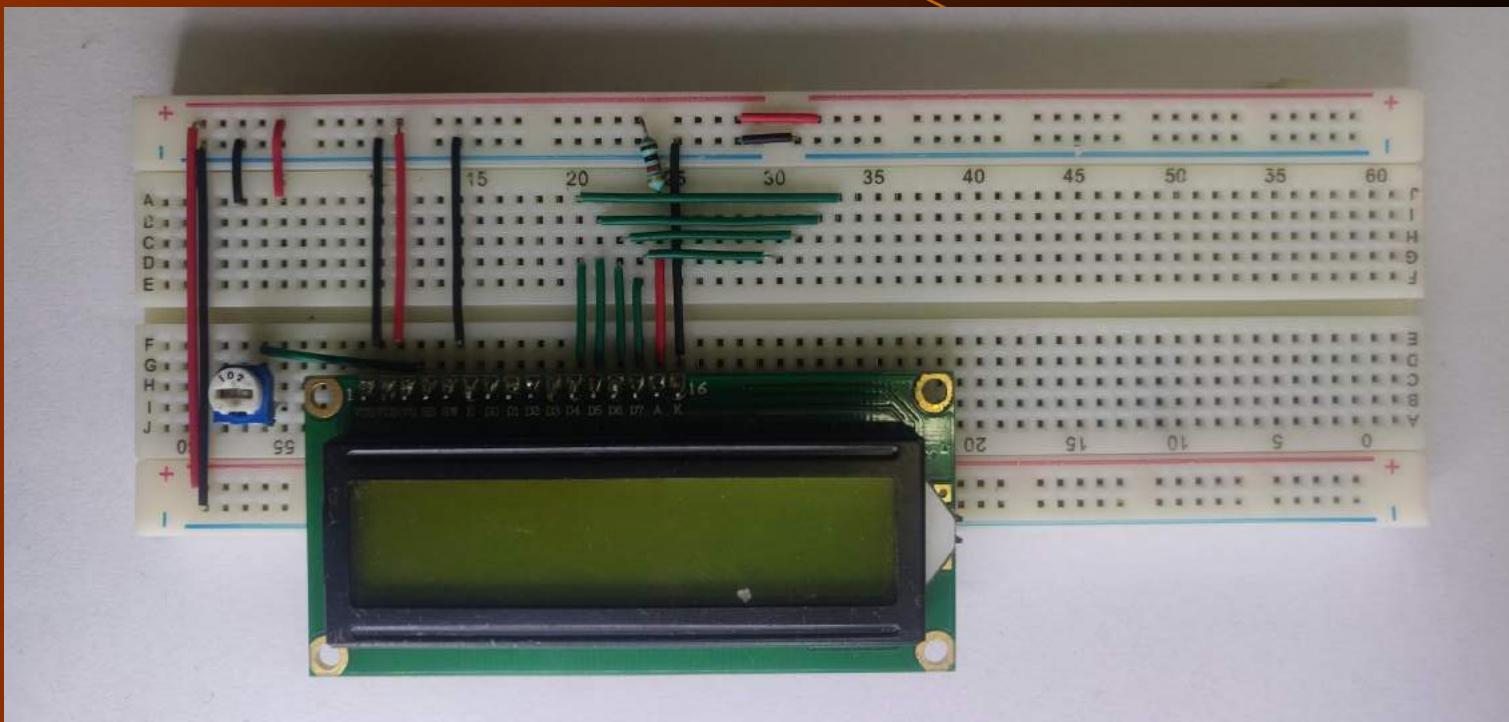
Circuits on Breadboard



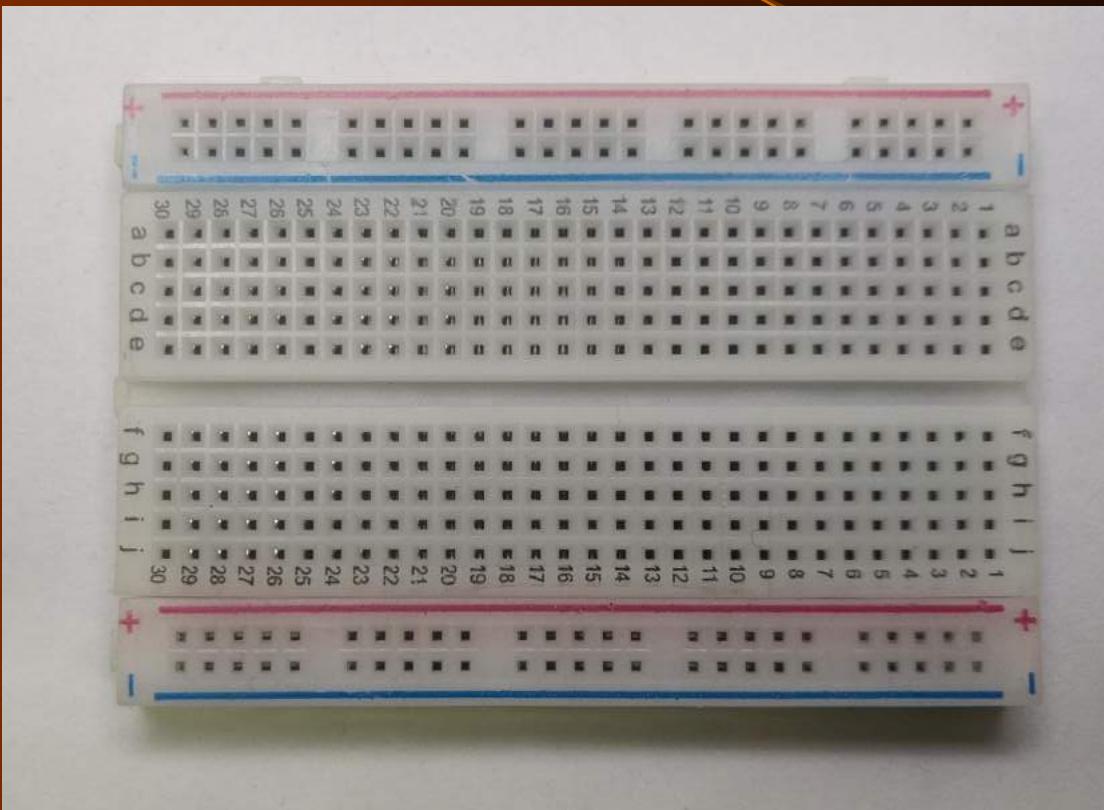
Circuits on Breadboard



Circuits on Breadboard



BreadBoard



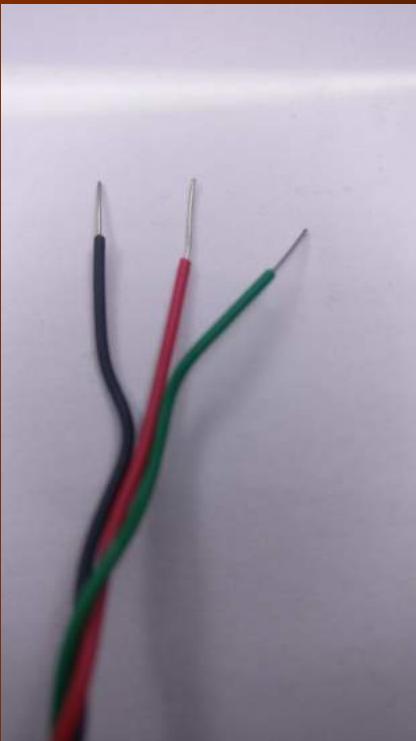
BreadBoard Internal View



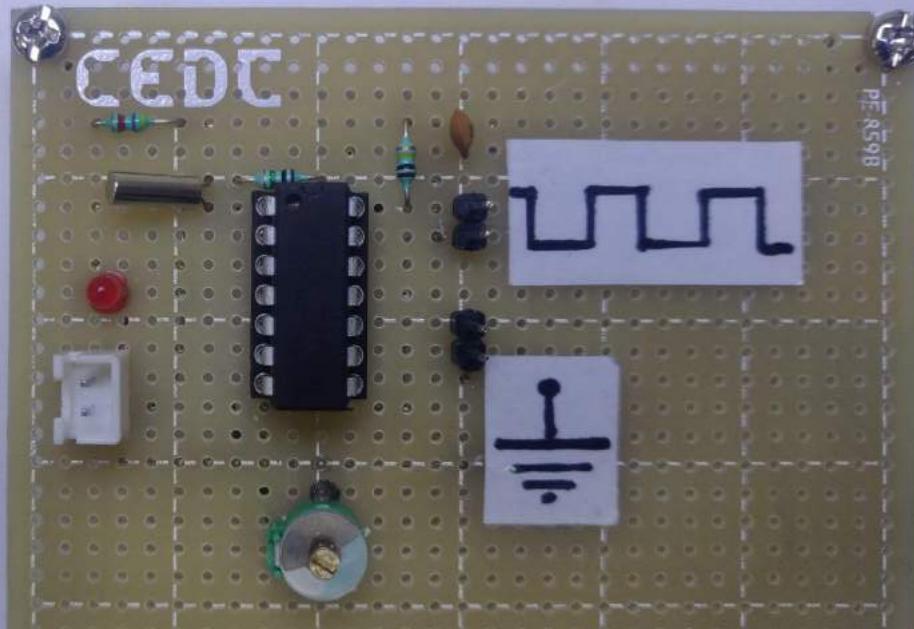
Tools for Prototyping on BreadBoard



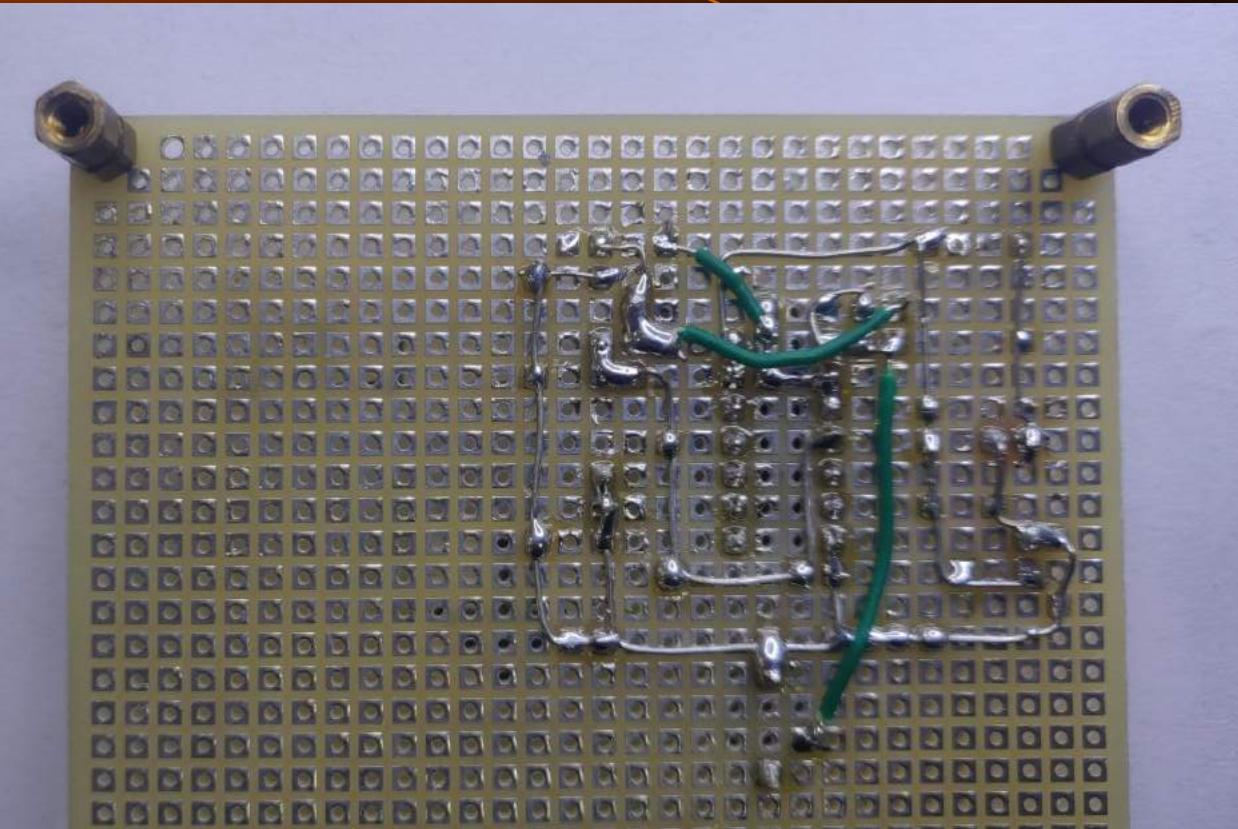
Single Strand Wires



Circuit on ZeroBoard



ZeroBoard Solder Side



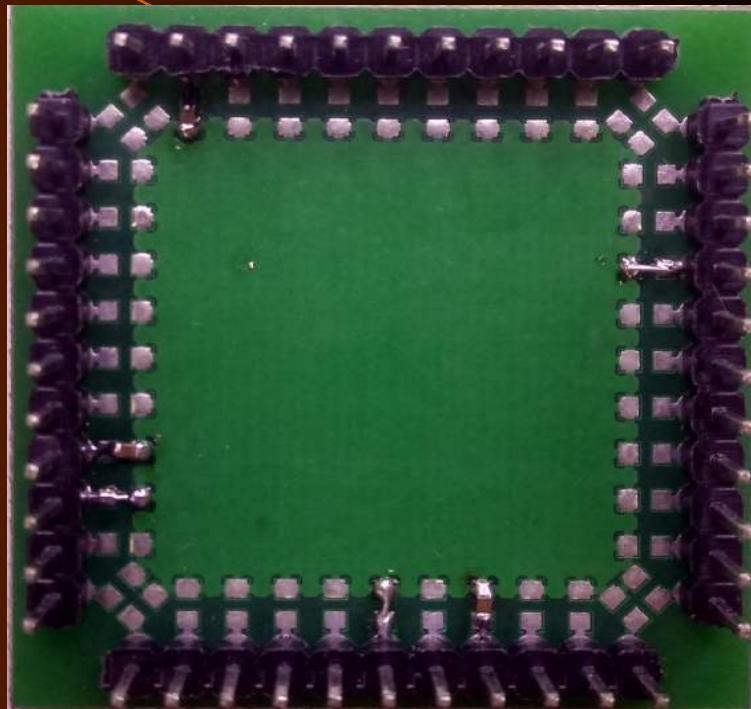
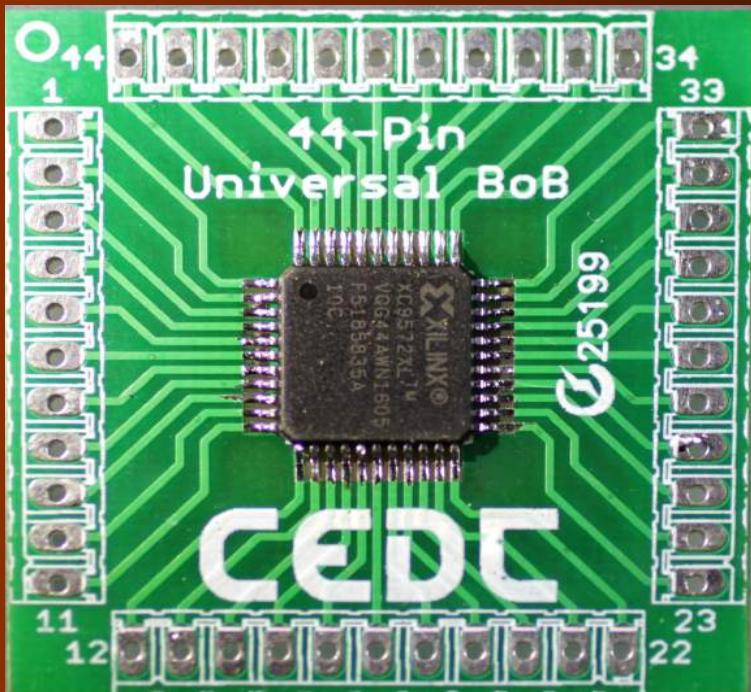
Tools for Prototyping on Zeroboard



Tools for Prototyping on Zeroboard

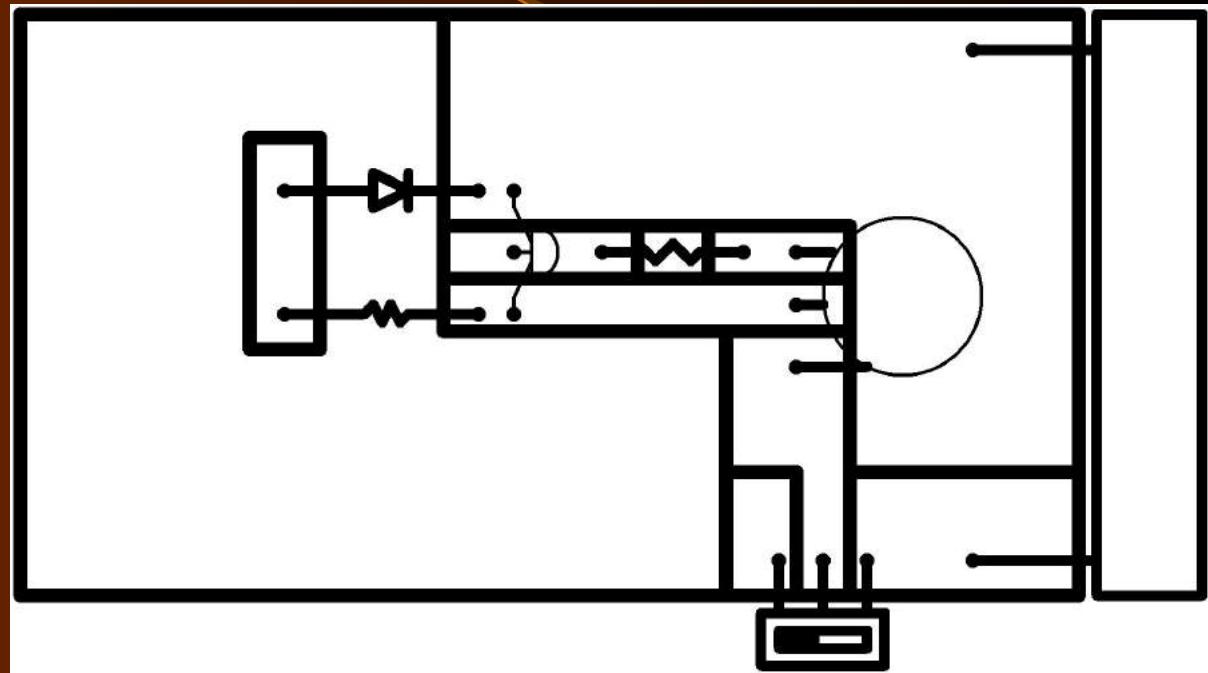
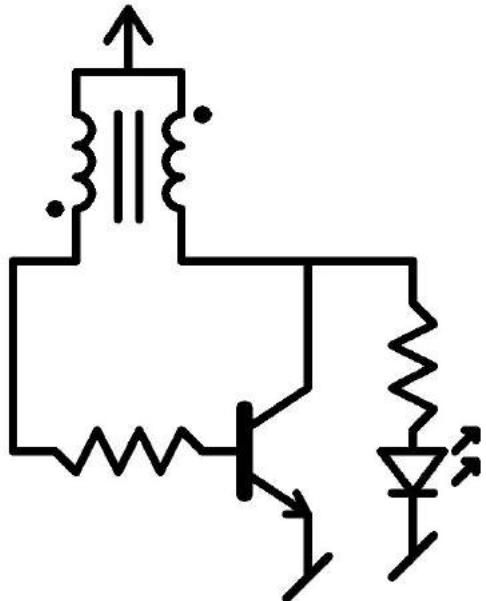


Break out Board (BOB)

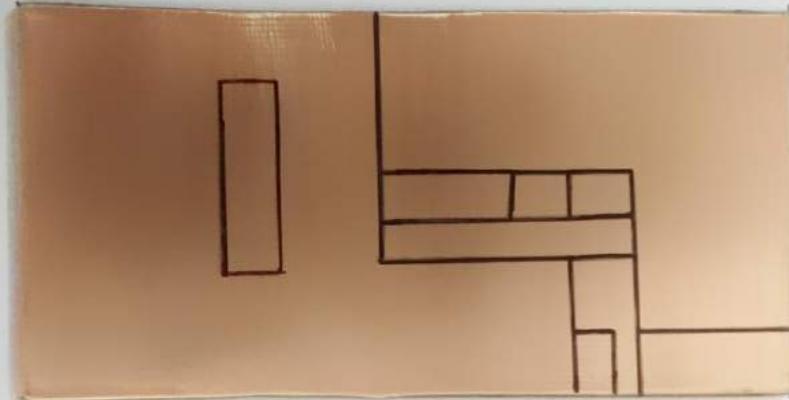


Manhattan Style

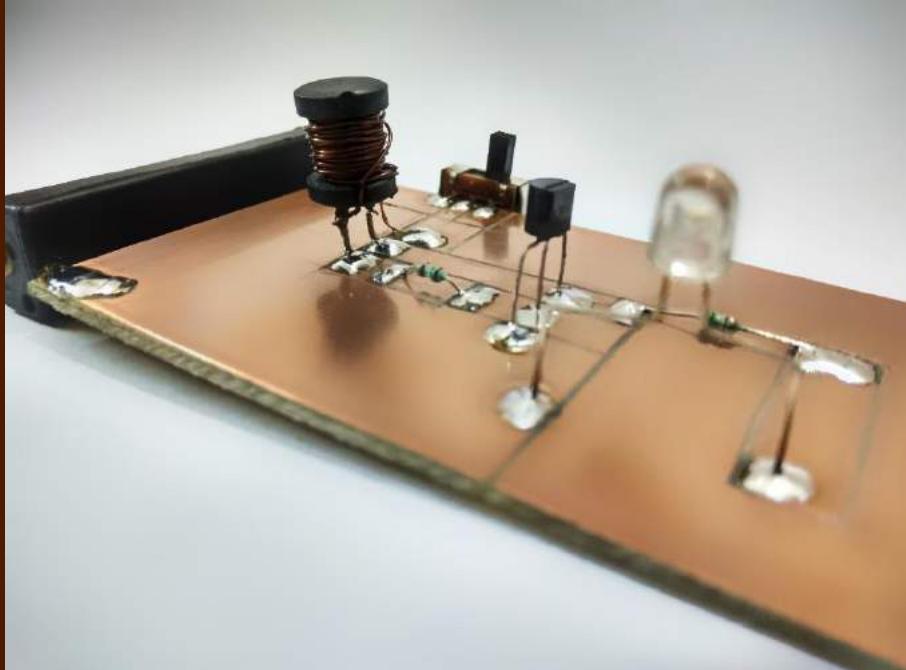
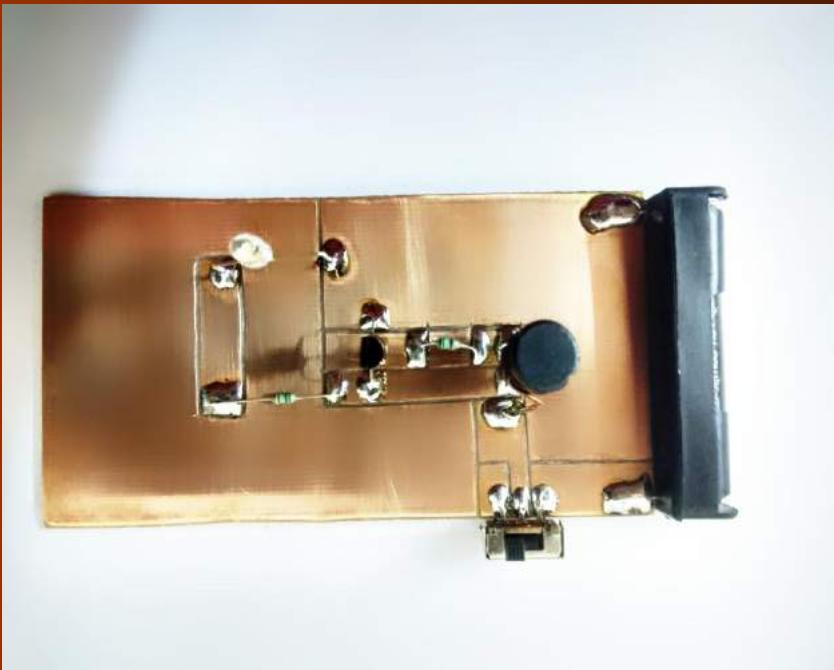
Joule Thief



PCB Preparation



Soldered PCB

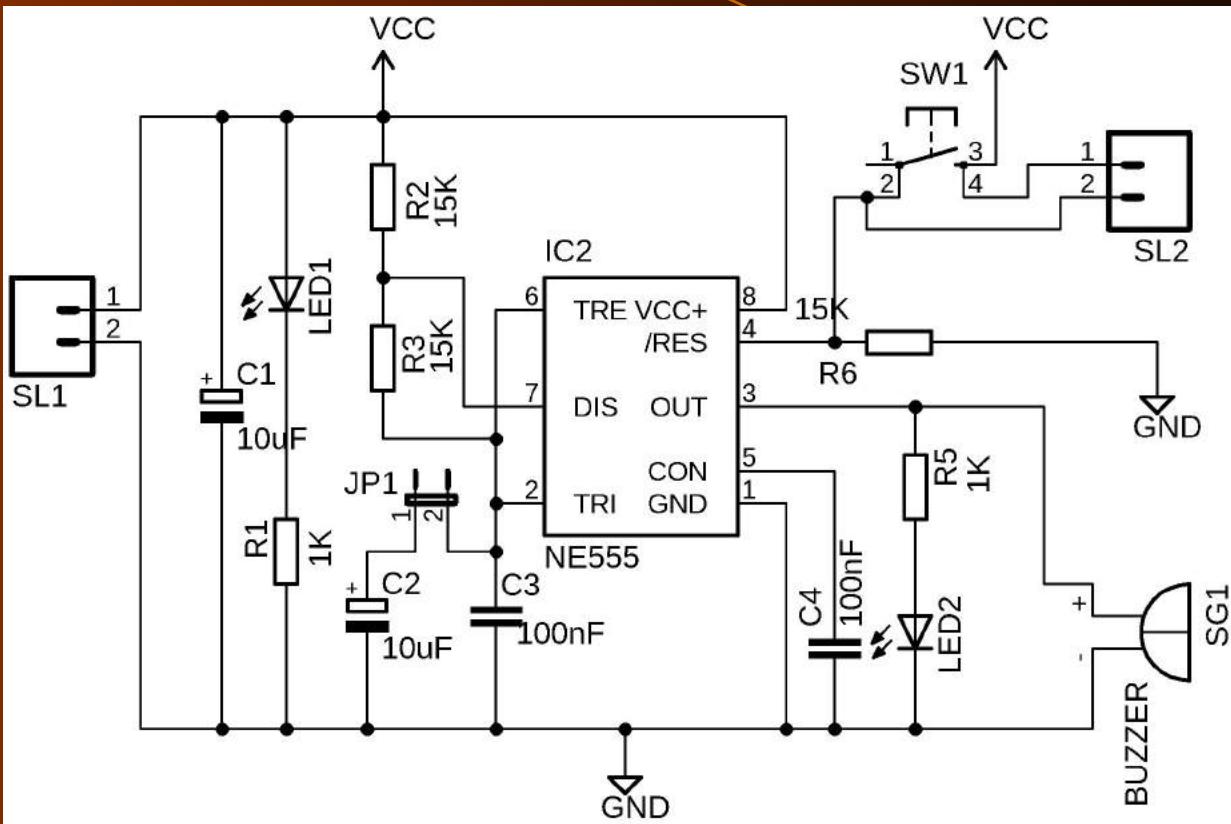


Custom PCB Fabrication

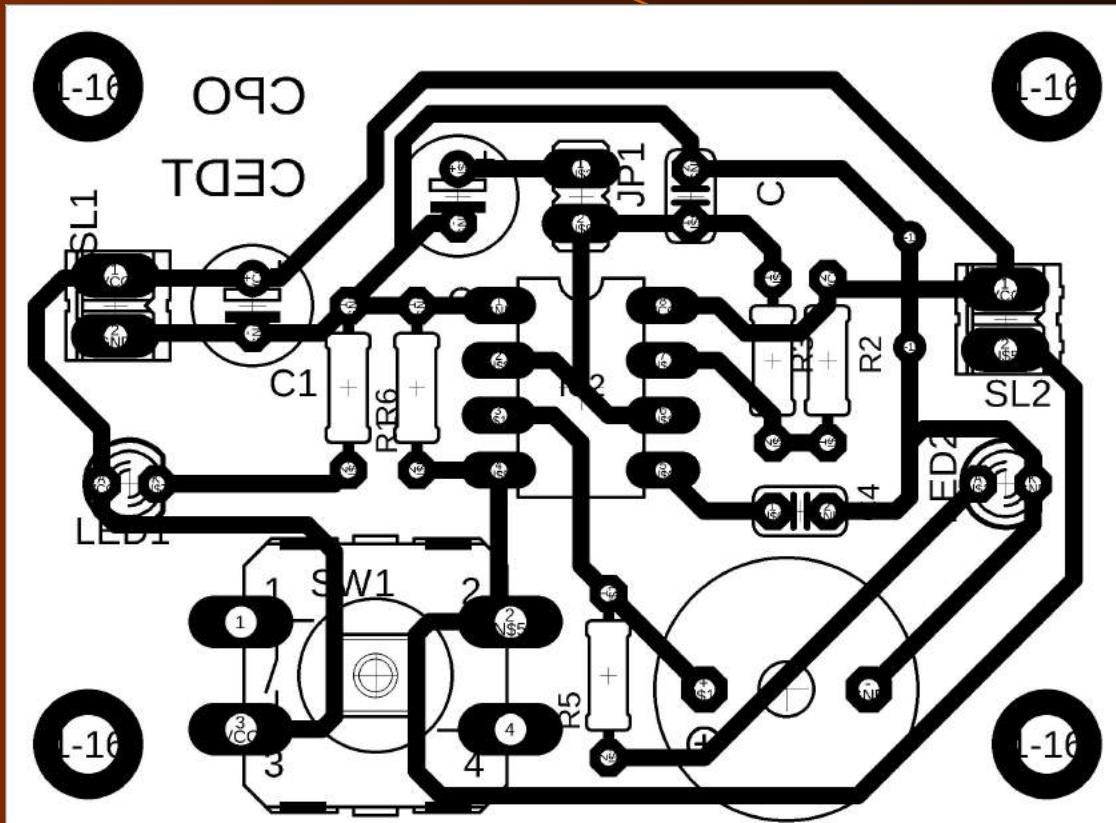
Toner Transfer Method

1. Copper Clad Cutting
2. Surface preparation: Sanding
3. Pressing the layout
4. Paper Removal
5. Removing Unwanted Paper with a Needle
6. Etching
7. Scrubbing
8. Acrylic Spray
9. Drilling
10. Soldering

Schematic Capture



Board Layout



Copper Clad Cutting

- Using Hacksaw : This is a manual method of cutting the copper clad in which the bigger copper sheet is aligned at an appropriate position with the help of supports and then is cut using a hacksaw. This method suffers from few limitations i.e. we can cut only limited geometrical shapes such as rectangular(non-curved), etc.
- Using CNC machine : CNC is a Computer Numeric Control machine. It executes pre-programmed sequences of machine control commands. Any required shape of PCB can be designed in software associated with the CNC machine. The CNC machine then cuts accordingly. Example of such a CNC machine is ShopBot.

CNC Machine



Sanding



Raw Copper
Clad

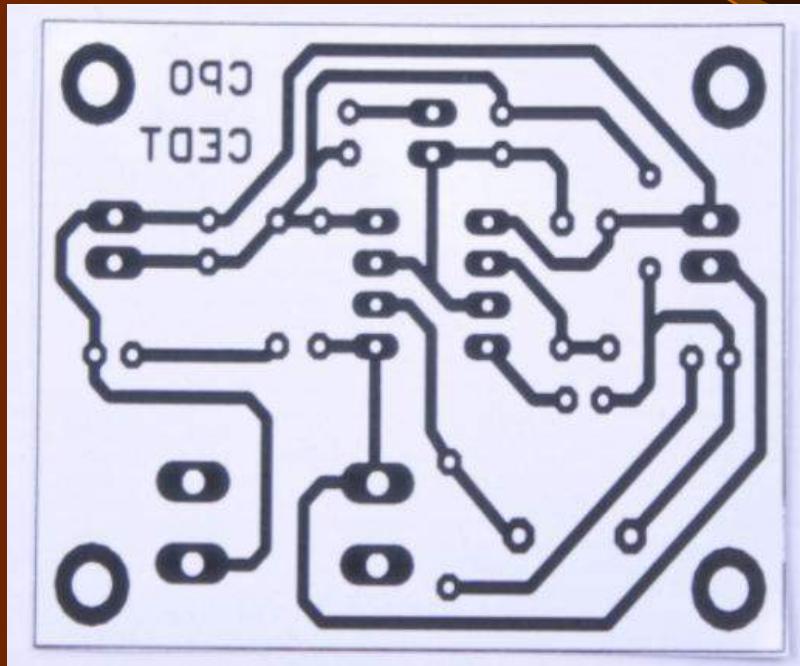


Sandpapering in
Progress



Prepared Copper
Clad

Printed Board Layout



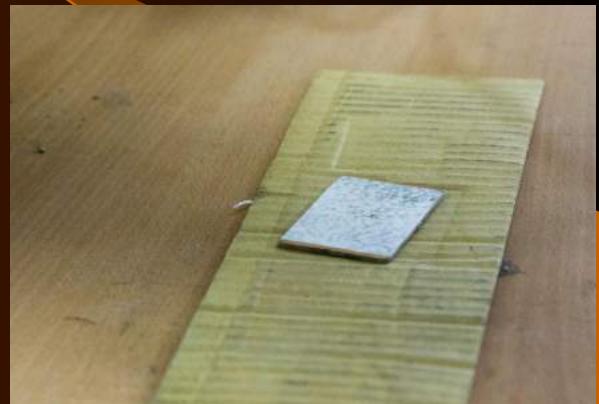
Pressing



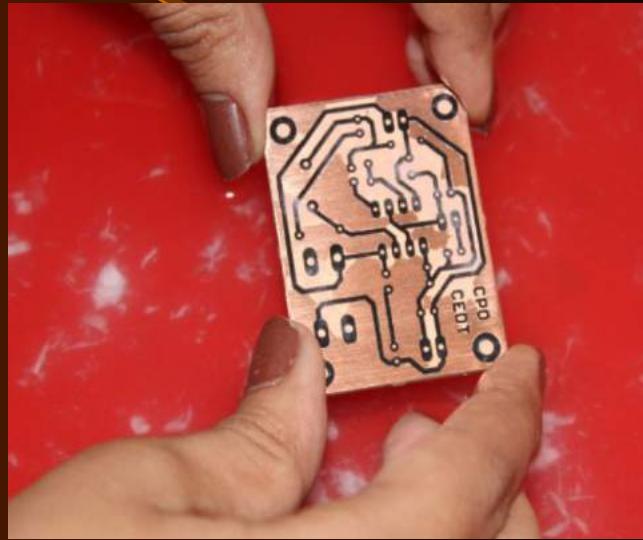
Pressing In Progress



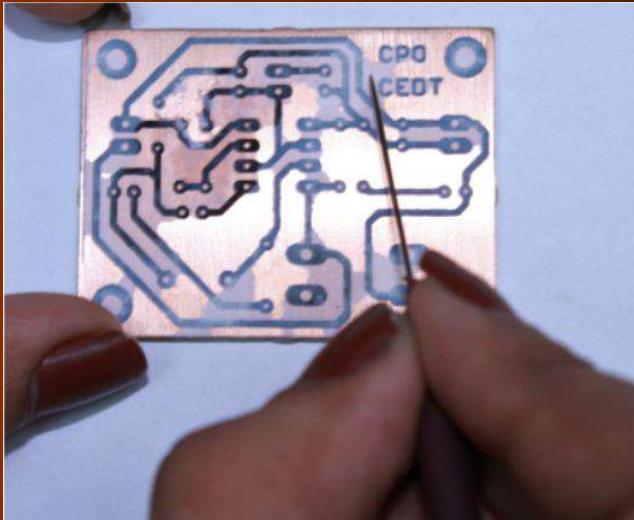
Pressed Board



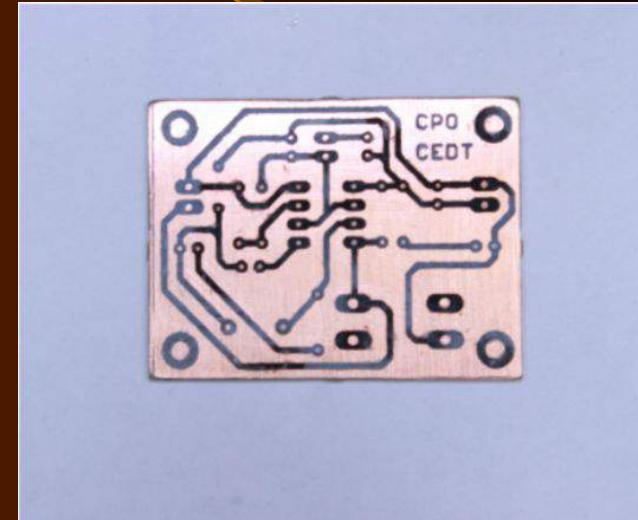
Paper Removal



Removing Unwanted Paper with a Needle



Removing in Progress



Board after Removing all
the Paper

Etching



Equations:



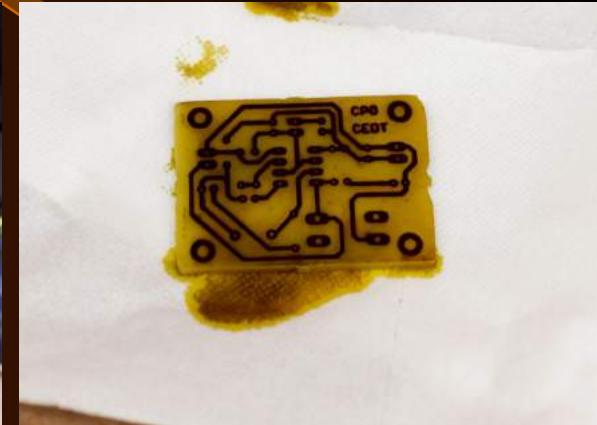
Etching



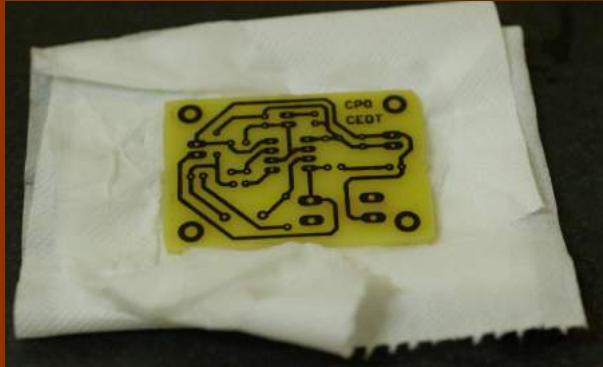
Etching in
Progress



Completely Etched Board



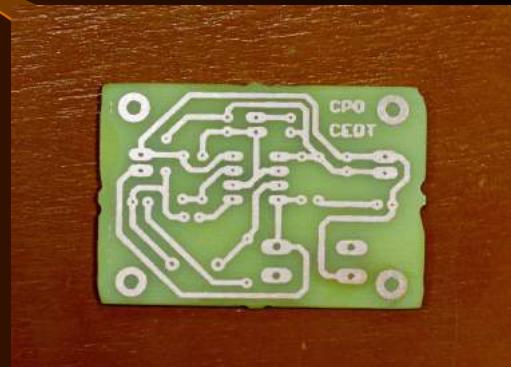
Scrubbing



Board Before
Scrubbing



Scrubbing In
Progress



Board After
Scrubbing

Acrylic Spray



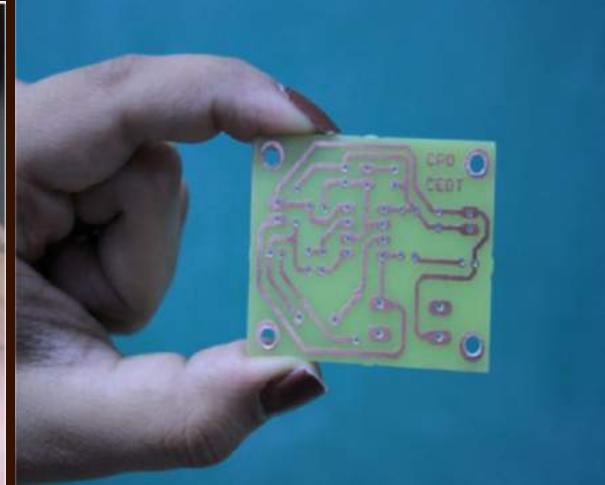
Drilling



Drilling
mounting holes

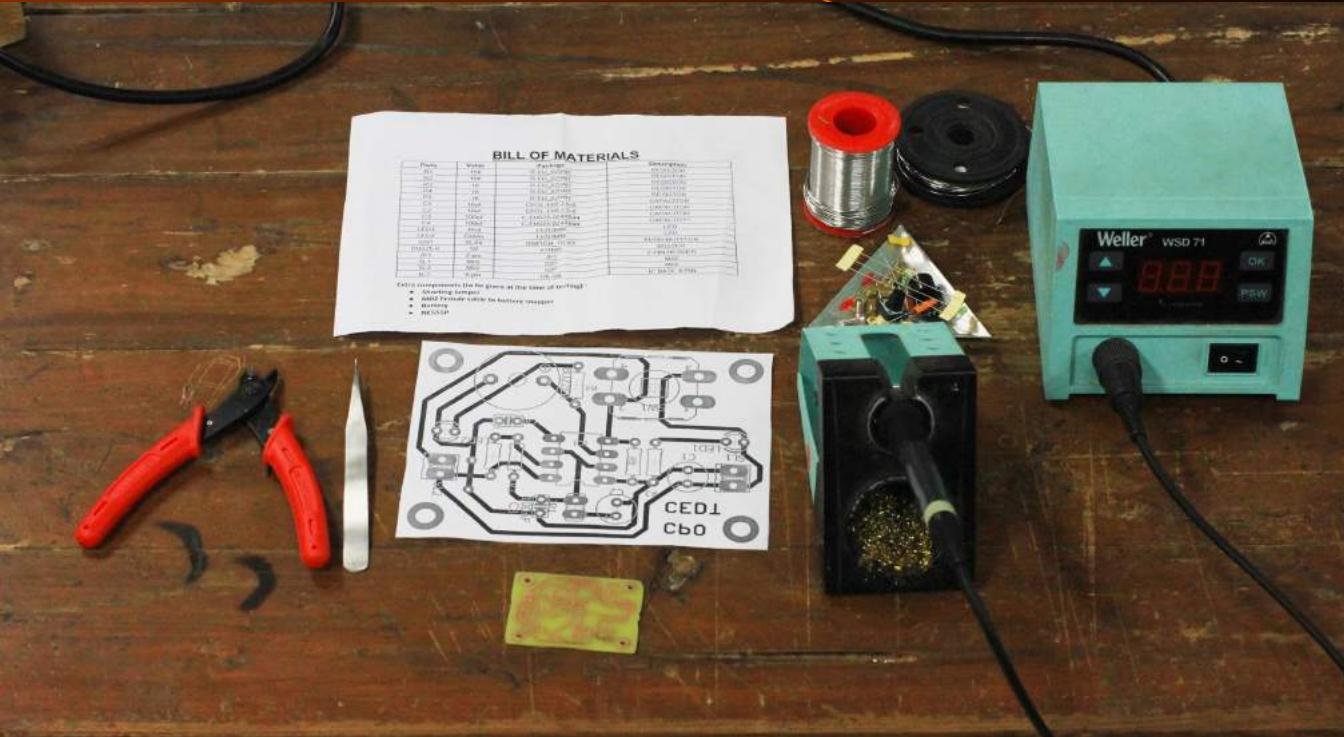


Drilling in
Progress

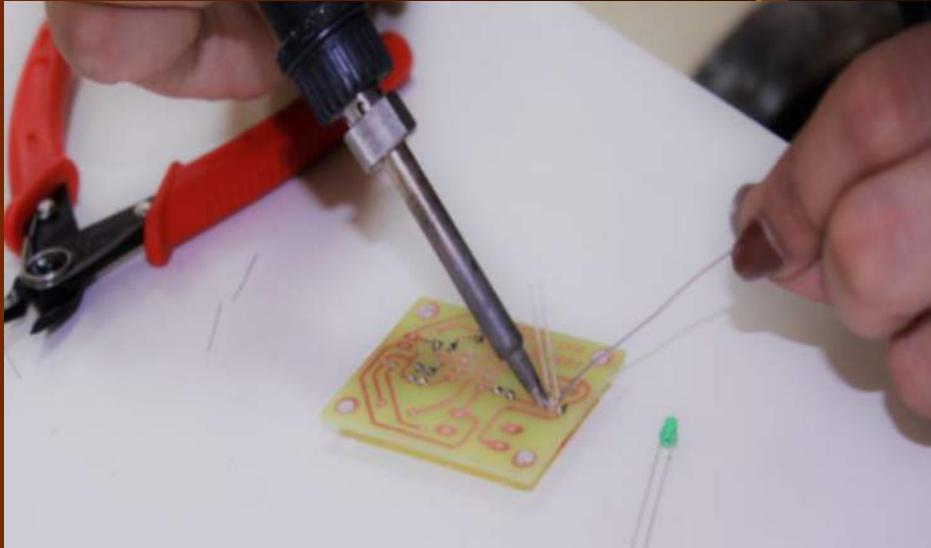


Board After
Drilling

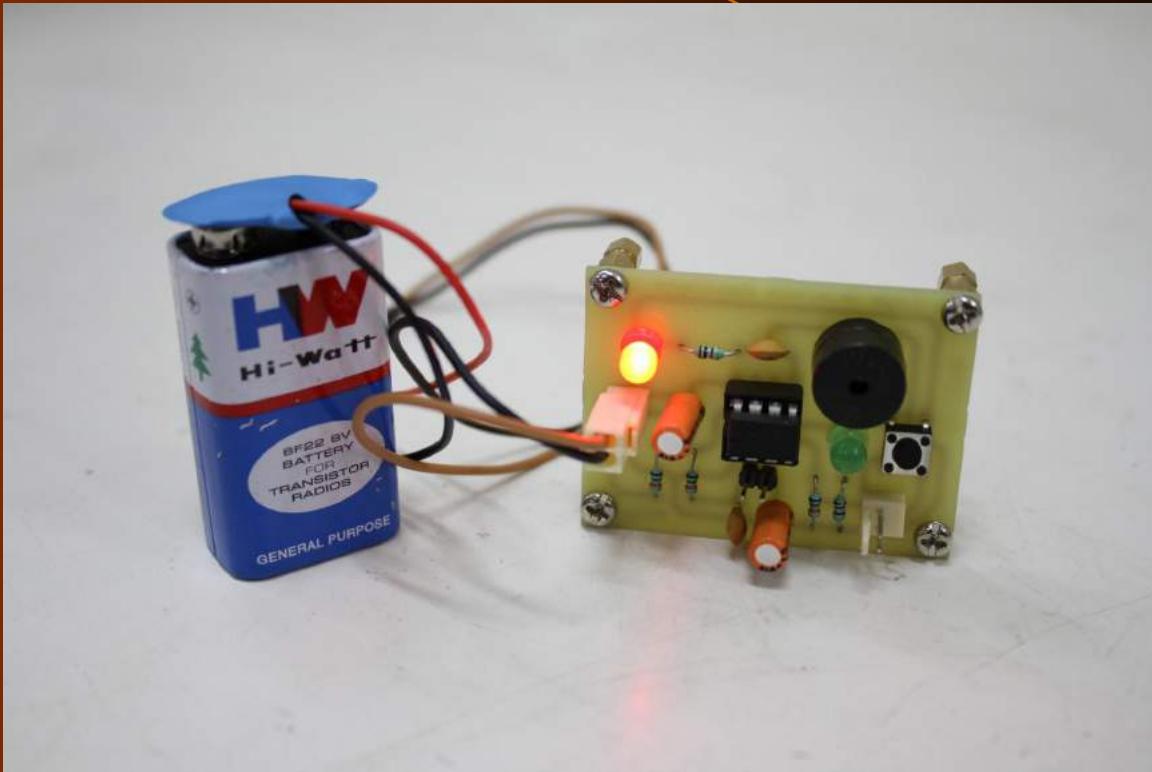
Soldering Setup



Soldering



Final PCB





Thank you!

Introduction to Embedded System Design

Single Purpose Computers

Dhananjay V. Gadre
Associate Professor
ECE Division
Netaji Subhas University of
Technology, New Delhi

Badri Subudhi
Assistant Professor
Electrical Engineering
Department
Indian Institute of Technology,
Jammu

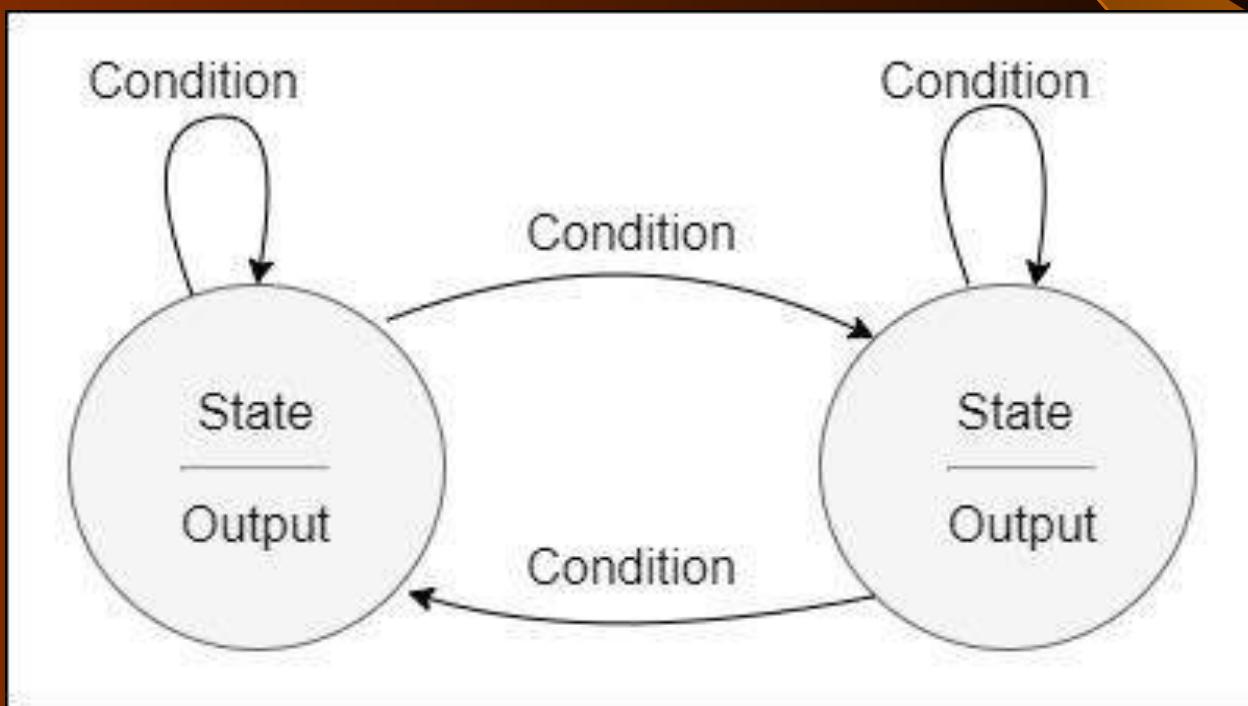
Finite State Machine(FSM)

- The tasks of a digital circuits can be arranged in sequences as per need, these are called states. In each state the machine performs certain tasks.
- When a certain machine has a characteristics of switching into other states in accordance to its inputs, past status or clock signals, these class of machines are called Finite State Machines.
- Also known as Finite Automata, Sequencer or Control Path.

Types of FSM

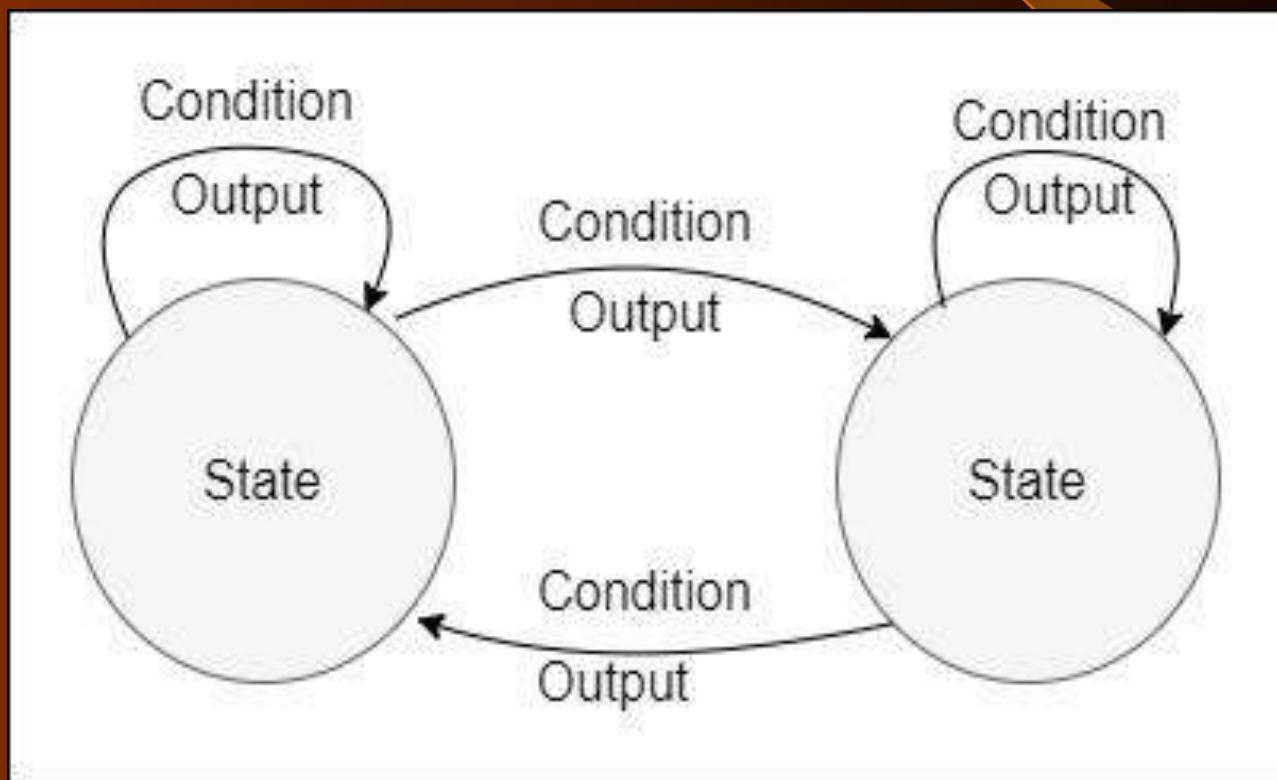
- MOORE MACHINE

A machine whose output depends only on present state.



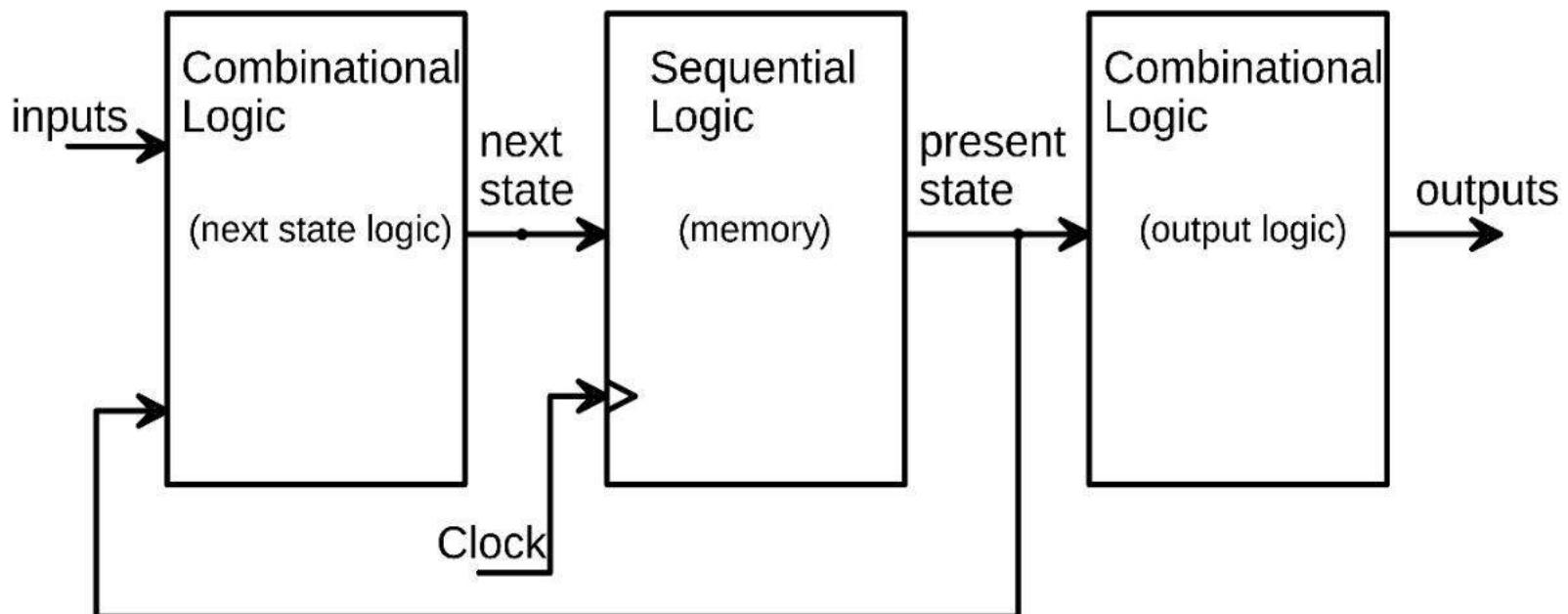
• MEALY MACHINE

A machine whose output depends on present state as well as current input.

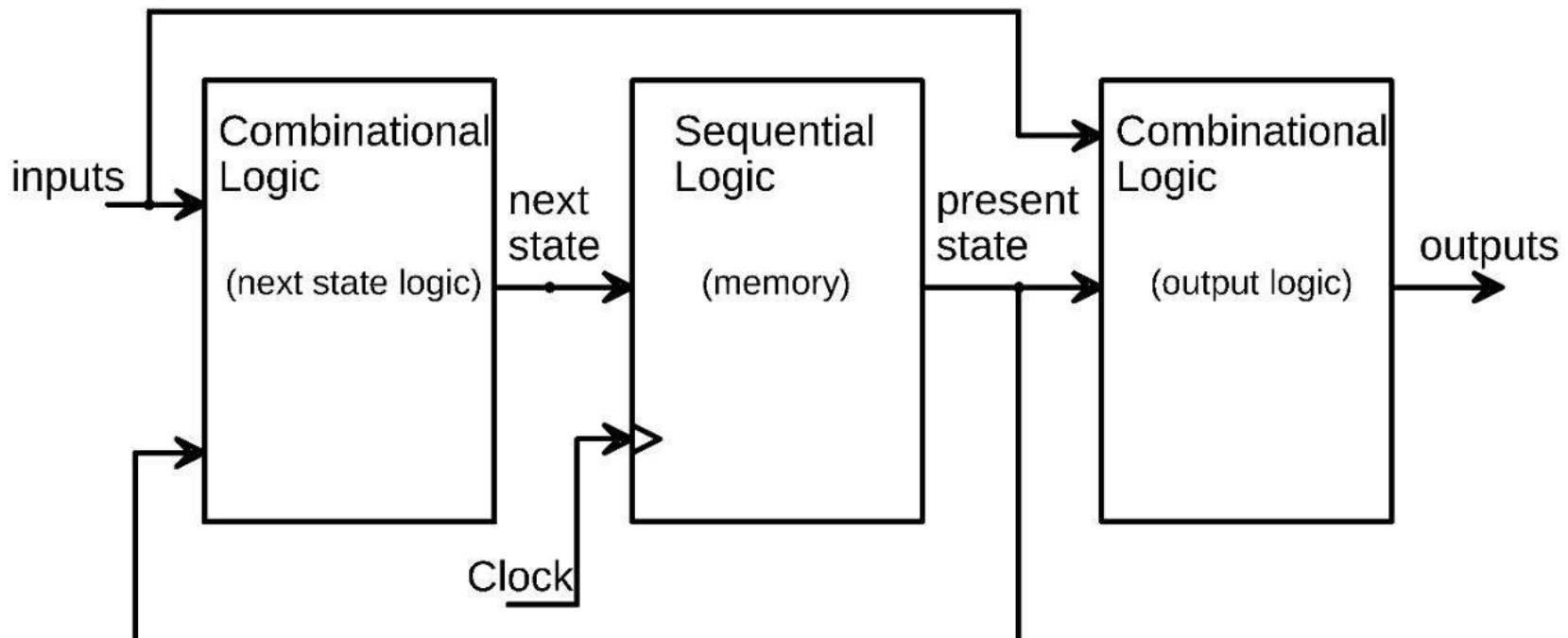


Representation as a 3 box model

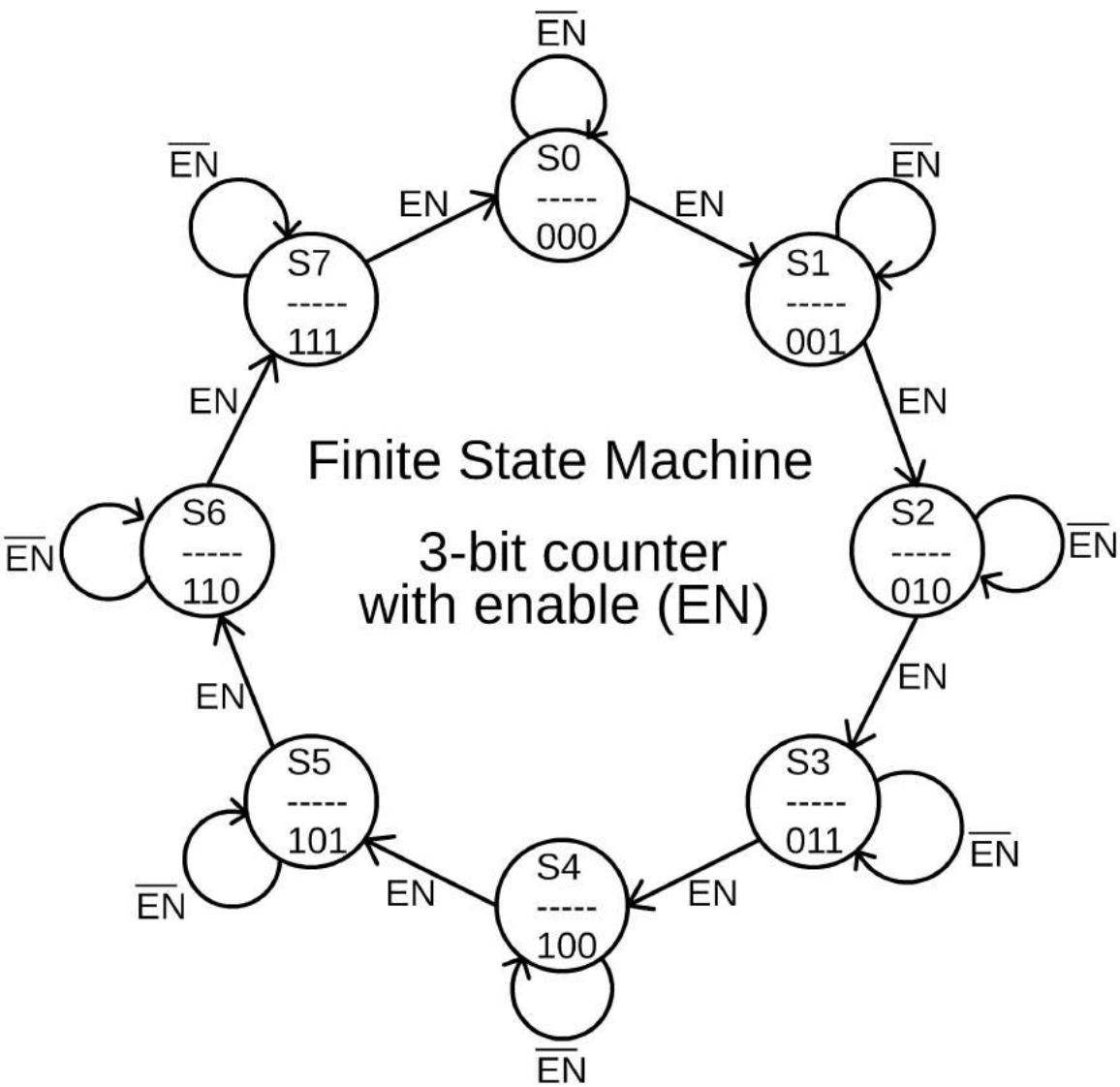
Moore Machine



Mealy Machine



Example Of FSM



VHDL Code:

```
entity Counter_FSM is
    Port ( clk,enable : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (2
downto 0));
end Counter_FSM;
architecture Behavioral of Counter_FSM is
signal state : integer range 0 to 7:=0;
begin
    process(clk,enable,state)
    begin
        if rising_edge(clk) then
            case state is
                when 0|1|2|3|4|5|6=>
                    if enable='1' then
                        state<=state+1;
                    end if;
                when others=> state<=0;
            end case;
        end if;
    end process;
```

VHDL Code Continued:

```
process(state)
begin
    case state is
        when 0=> output<="000";
        when 1=> output<="001";
        when 2=> output<="010";
        when 3=> output<="011";
        when 4=> output<="100";
        when 5=> output<="101";
        when 6=> output<="110";
        when 7=> output<="111";
    end case;
end process;
end Behavioral;
```

DATAPATH

Datapath refers to a functional block of a machine consisting of all elements used for arithmetic, logical and data transfer operations which it needs to perform in each state.

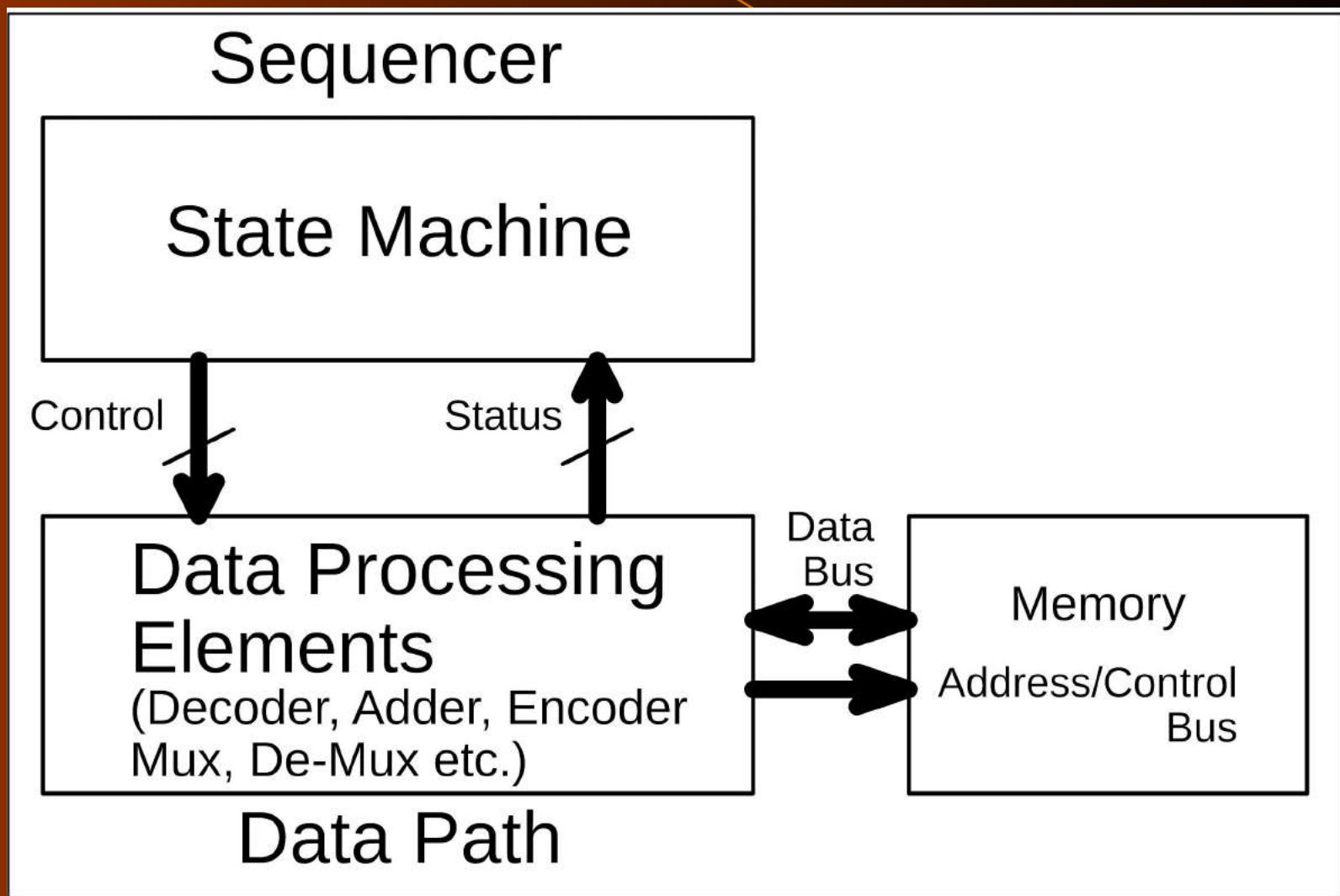
FSM-D

- When an FSM sequences operations into a datapath, together they are referred as Finite State Machine with Datapath.
- They are very useful as they are flexible enough to implement any logic circuit. In fact many embedded system can be designed using this technique.

FSM-D

- FSM sends control signals to the datapath, as an instruction for it to perform certain task needed in that state.
- In return datapath can send various status signals back to FSM (Example- data in process, processing done, wait etc.). Based on these signals FSM then decides in which state it has to proceed next.

BLOCK DIAGRAM OF FSMD

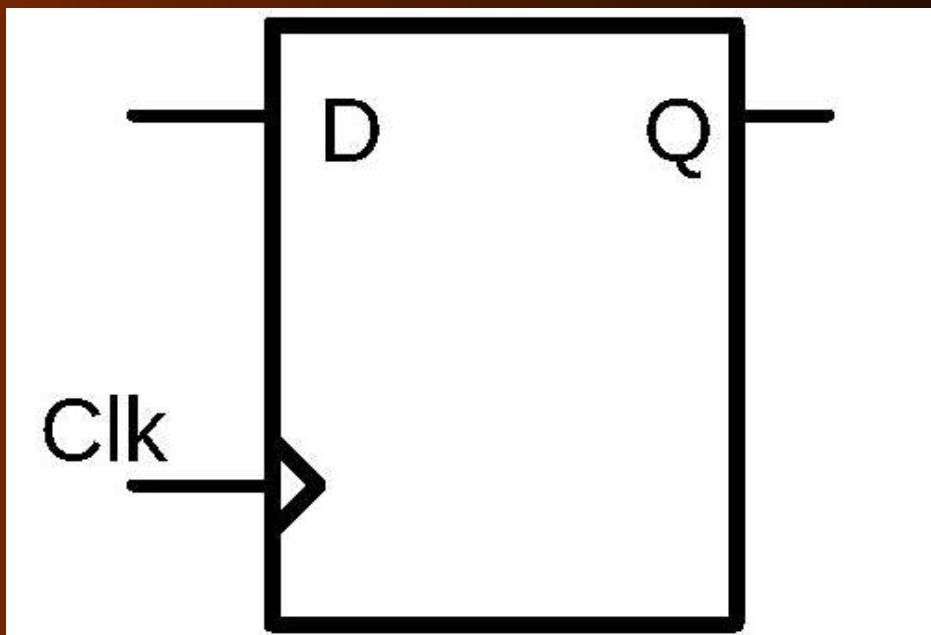


Elements Of Datapath

1. Latch/Flip Flop
2. Flip-Flop with Enable
3. Tri State Buffer
4. Comparators
5. Encoder
6. Decoder
7. Multiplexer
8. Counters
9. Arithmetic and Logical Unit(ALU)
10. Clock Divider

Flip Flop

- Flip flops are edge sensitive digital circuits.
- Depending on particular edge (rising or falling), it captures data.



VHDL Code:

```
entity Flip_Flop is
    Port ( clk,D : in STD_LOGIC;
           Q,Q_bar : buffer STD_LOGIC);
end Flip_Flop;
```

```
architecture Behavioral of Flip_Flop is
begin
```

```
    process(clk,D)
```

```
    begin
```

```
        if rising_edge(clk) then
```

```
            Q<=D;
```

```
        end if;
```

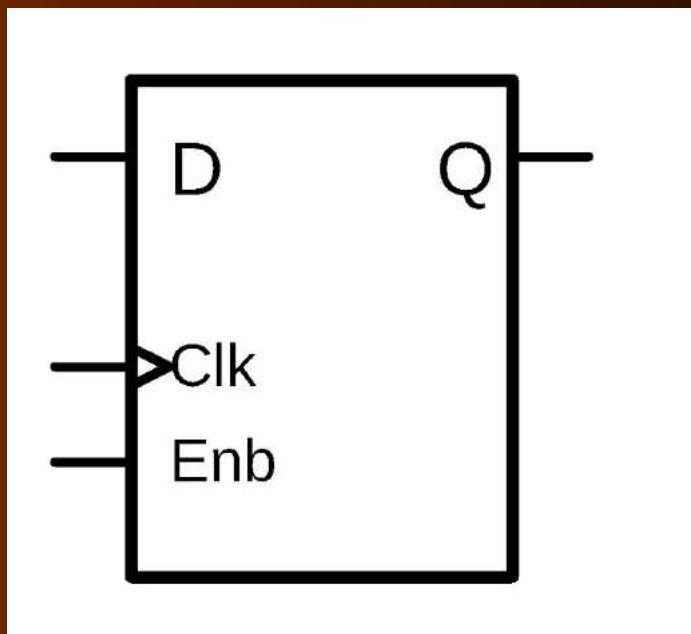
```
    end process;
```

```
    Q_bar<= not Q;
```

```
end Behavioral;
```

Controlled Latch

- Latch sets, resets the output depending on the input .
- It performs latch operation whenever controlling signal (Enable) is given to latch.



VHDL Code:

```
entity Controlled_Latch is
port(enable,clk,D: in std_logic;
      Q, Q_bar: buffer std_logic);
end Controlled_Latch;
```

```
architecture Behavioral of Controlled_Latch is
begin
```

```
  process(clk,enable)
  begin
    if enable='1' then
      if clk='1' then
```

```
        Q<=D;
```

```
      end if;
```

```
    end if;
```

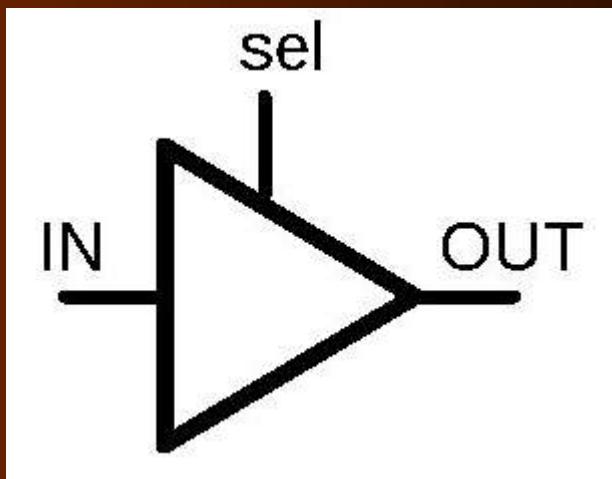
```
  end process;
```

```
  Q_bar<=not Q;
```

```
end Behavioral;
```

Tri State Buffer

- It is almost similar to a buffer but has three ports instead of two, the third pin is called enable which controls its operation.
- If enable is at high logic, input will be copied to output otherwise output will be at high impedance state.



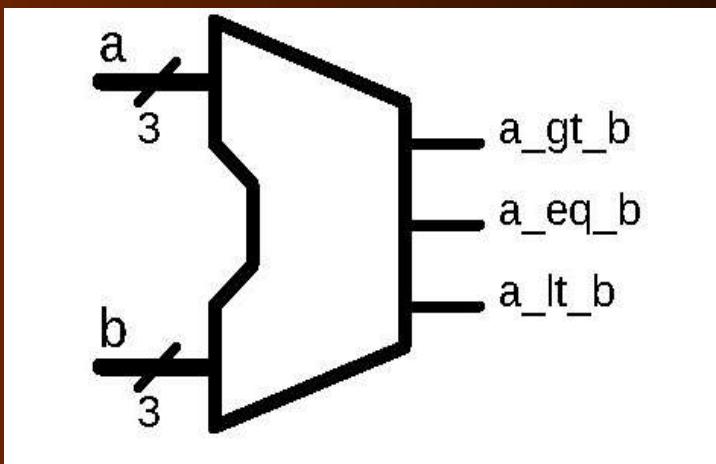
VHDL Code:

```
entity Tri_State_Buffer is
    Port ( data,sel : in STD_LOGIC;
           output : out STD_LOGIC);
end Tri_State_Buffer;
```

```
architecture Behavioral of Tri_State_Buffer is
begin
    output<=data when sel='1' else
        'Z';
end Behavioral;
```

Comparators

- Comparator is a device which compares two input signals (A and B), and based on that it gives output as $A > B$, $A < B$ and $A = B$.



VHDL Code:

```
entity Comparator is
port(a,b: in std_logic_vector(3 downto 0);
      high,equal,less : out std_logic);
end Comparator;
```

```
architecture Behavioral of Comparator is
begin
```

```
process(a,b)
```

```
begin
```

```
    high<='0'; equal<='0'; less<='0';
```

```
    if a>b      then    high<='1';
```

```
    elsif (a=b) then    equal<='1';
```

```
    elsif a<b     then    less<='1';
```

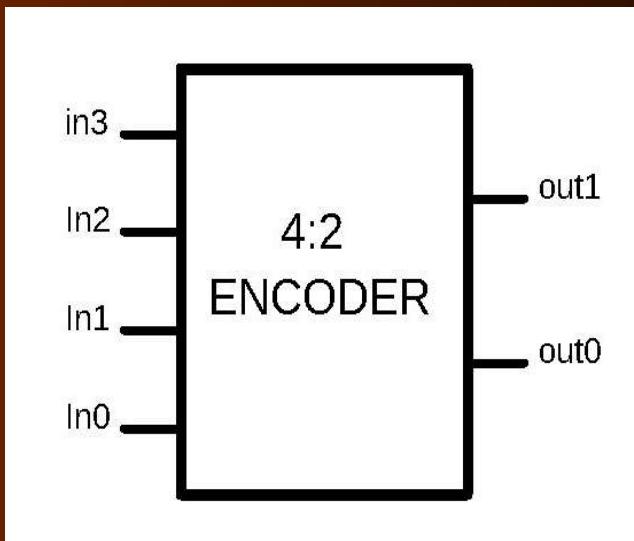
```
    end if;
```

```
end process;
```

```
end Behavioral;
```

Encoder

- Encoder converts 2^n bit input to unique output of n-bits.
- Only one bit in input is HIGH. Encoder output is binary equivalent of position of this HIGH.

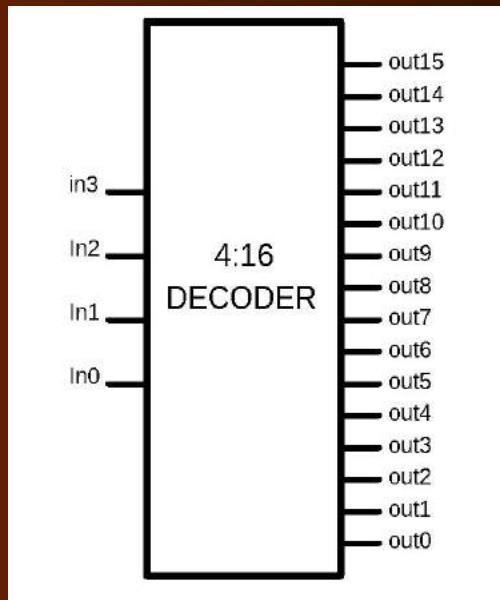


VHDL Code:

```
entity Encoder_8X3 is
    port( input: in std_logic_vector(7 downto 0);
          output : out std_logic_vector(7 downto 0));
end Encoder_8X3;
architecture Behavioral of Encoder_8X3 is
begin
    process(input)
    begin
        case(input) is
            when "0000_0001" => output<="000";
            when "0000_0010" => output<="001";
            when "0000_0100" => output<="010";
            when "0000_1000" => output<="011";
            when "0001_0000" => output<="100";
            when "0010_0000" => output<="101";
            when "0100_0000" => output<="110";
            when "1000_0000" => output<="111";
            when others=> output<="ZZZ";
        end case;
    end process;
```

Decoder

- Decoder converts a n-bit binary input to unique output of 2^n bits.
- Position of only one logic ‘HIGH’ bit in output is the decimal equivalent of n-bit binary input.

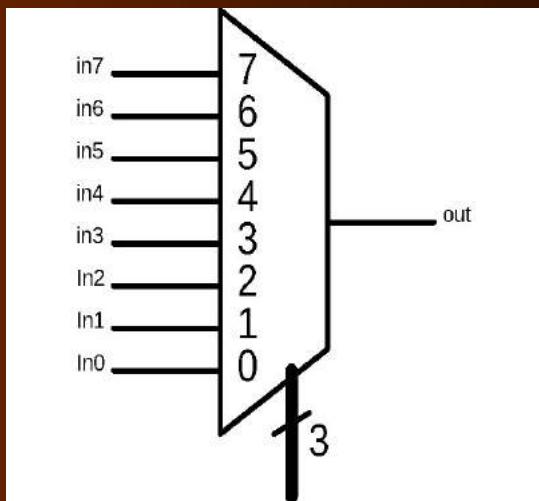


VHDL Code:

```
entity Decoder_3X8 is
    port(sel:in std_logic_vector(2 downto 0);
         output: out std_logic_vector(7 downto 0));
end Decoder_3X8;
architecture Behavioral of Decoder_3X8 is
begin
process(sel)
begin
    case sel is
        when "000" => output <= "00000001";
        when "001" => output <= "00000010";
        when "010" => output <= "00000100";
        when "011" => output <= "00001000";
        when "100" => output <= "00010000";
        when "101" => output <= "00100000";
        when "110" => output <= "01000000";
        when "111" => output <= "10000000";
        when others => output <= "00000000";
    end case;
end process;
end Behavioral;
```

Multiplexer

- Multiplexer connects the output to one of the 2^n inputs based on n-bit select lines.



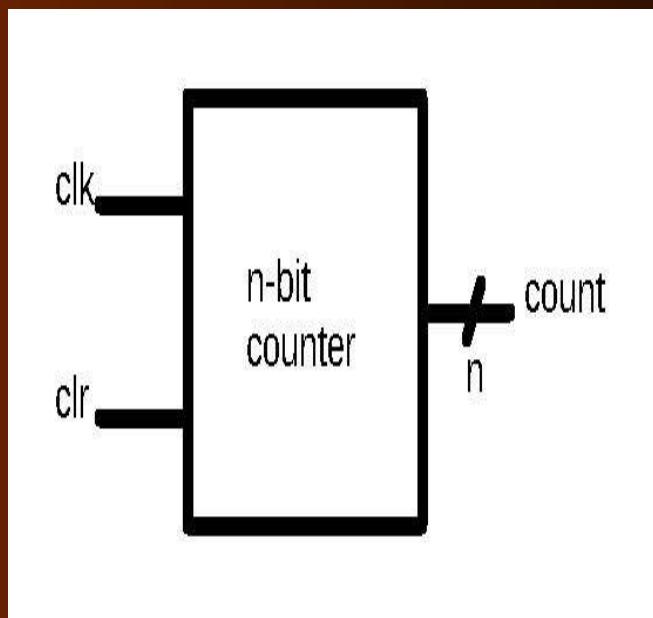
VHDL Code:

```
entity Mux_16X1 is
    Port ( input : in STD_LOGIC_VECTOR (15 downto 0);
           sel : in STD_LOGIC_VECTOR (3 downto 0);
           output : out STD_LOGIC);
end Mux_16X1;
```

```
architecture Behavioral of Mux_16X1 is
begin
    output<=input(conv_integer(unsigned(sel)));
end Behavioral;
```

Counter

- Counter increments or decrements it's value after a particular event is occurred (clock).

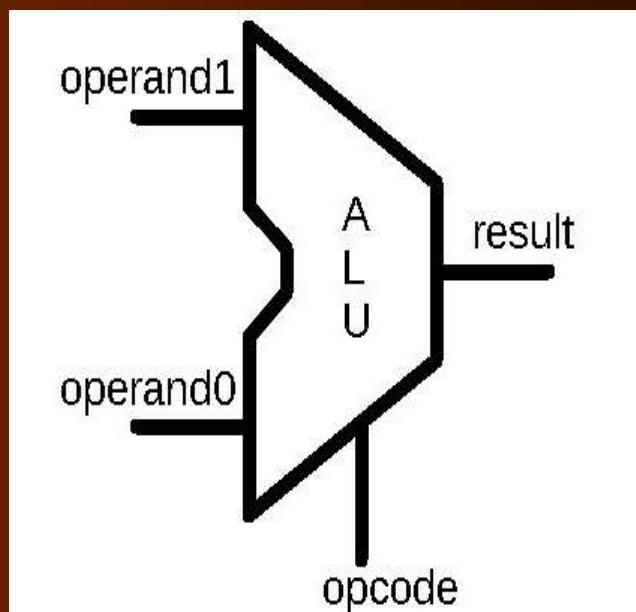


VHDL Code:

```
entity Counter is
    port(clk,reset,enable: in std_logic;
         count : buffer std_logic_vector(7 downto 0));
end Counter;
architecture Behavioral of Counter is
begin
    process(clk,reset,enable,count)
    begin
        if reset='1' then
            count<="00000000";
        elsif enable ='1' then
            if rising_edge(clk) then
                count<=count+1;
            end if;
        end if;
    end process;
end Behavioral;
```

Arithmetic and Logical Unit

- ALU is a basic building block which can be used to perform different arithmetic and logical operations on ‘operands’ depending upon value of ‘opcode’.

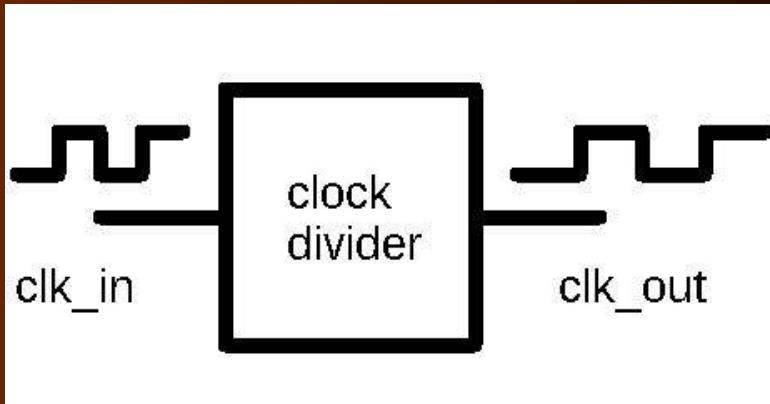


VHDL Code:

```
entity ALU is
    port( a,b : in std_logic_vector(3 downto 0);
          op: in std_logic_vector(2 downto 0);
          z: out std_logic_vector(3 downto 0));
end ALU;
architecture Behavioral of ALU is
begin
    process(a,b,op)
    begin
        case op is
            when "000"=> z<= a and b;
            when "001"=> z<= a or b;
            when "010"=> z<= not a;
            when "011"=> z<= a xor b;
            when "100"=> z<= a + b;
            when "101"=> z<= a - b;
            when "110"=> z<= a + 1;
            when "111"=> z<= a -1;
            when others=> NULL;
        end case;
    end process;
end Behavioral;
```

Clock Dividers

- Clock Divider is a circuit with the help of which we divide the on board system clock.



VHDL Code:

```
entity Clk_Divider is
port( sys_clk: in std_logic;
      divided_clk : out std_logic);
end Clk_Divider;
```

```
architecture Behavioral of Clk_Divider is
signal count: std_logic_vector(3 downto 0);
begin
  process(sys_clk,count)
  begin
    if rising_edge(sys_clk) then
      count<=count+1;
    end if;
  end process;
  divided_clk<=count(3);
end Behavioral;
```

Dual Dice Game

Problem Statement:

Design a game of rolling two dice using a switch & display the number on SSD.

Stage 1:

- a) User rolls the dice by pressing a switch.
- b) The number obtained is displayed on SSD when user leaves the switch.
- c) If the number obtained is :

$3/6/9 \rightarrow$ player wins. $2/8/11 \rightarrow$ player loses.

other \rightarrow the number obtained is stored. Goes to Stage 2.

Stage 2:

- a) User rolls the dice by again pressing the switch.
- b) The next number is displayed on the SSD.
- c) If the new number obtained is:

equal to the previously stored number then \rightarrow player wins

$3/6/9 \rightarrow$ player loses.

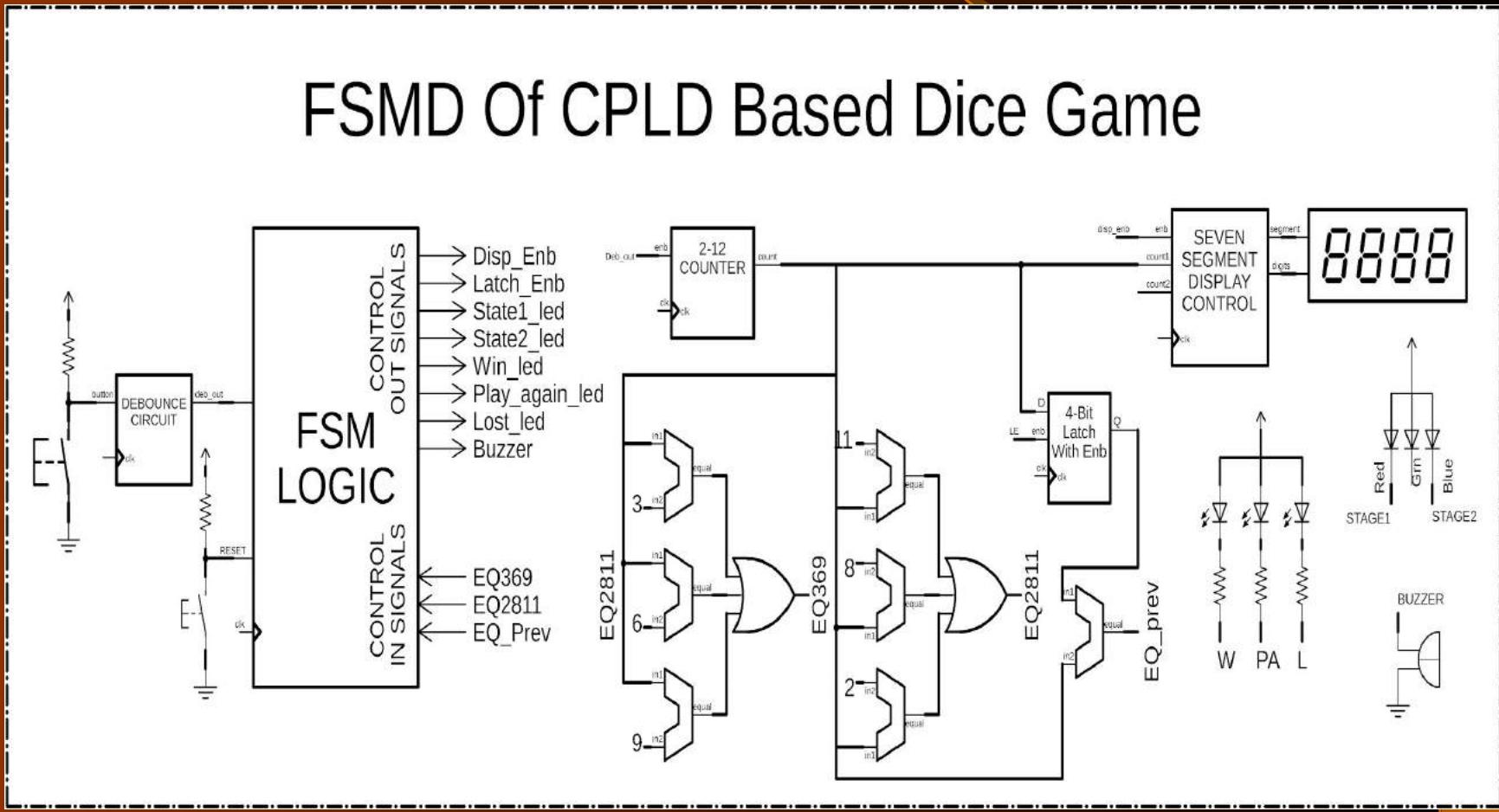
other \rightarrow the number obtained is stored. Goes to Stage 2.

Elements of Datapath in Dual Dice Game

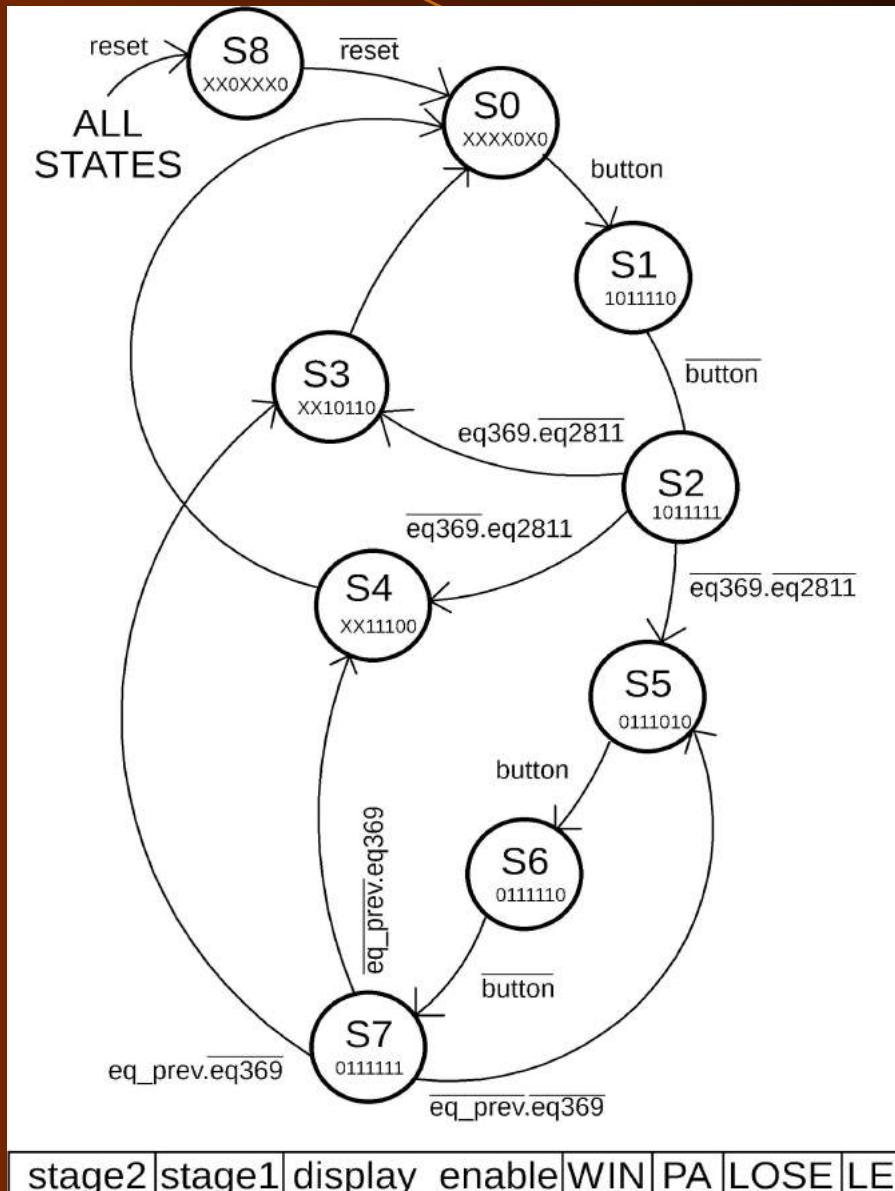
1. Debouncing Circuit
2. Counter
3. Comparators
4. Latch With Enable
5. Decoder for SSD

FSMD Of CPLD Based Dice Game

FSMD Of CPLD Based Dice Game



FSM of Dice Game



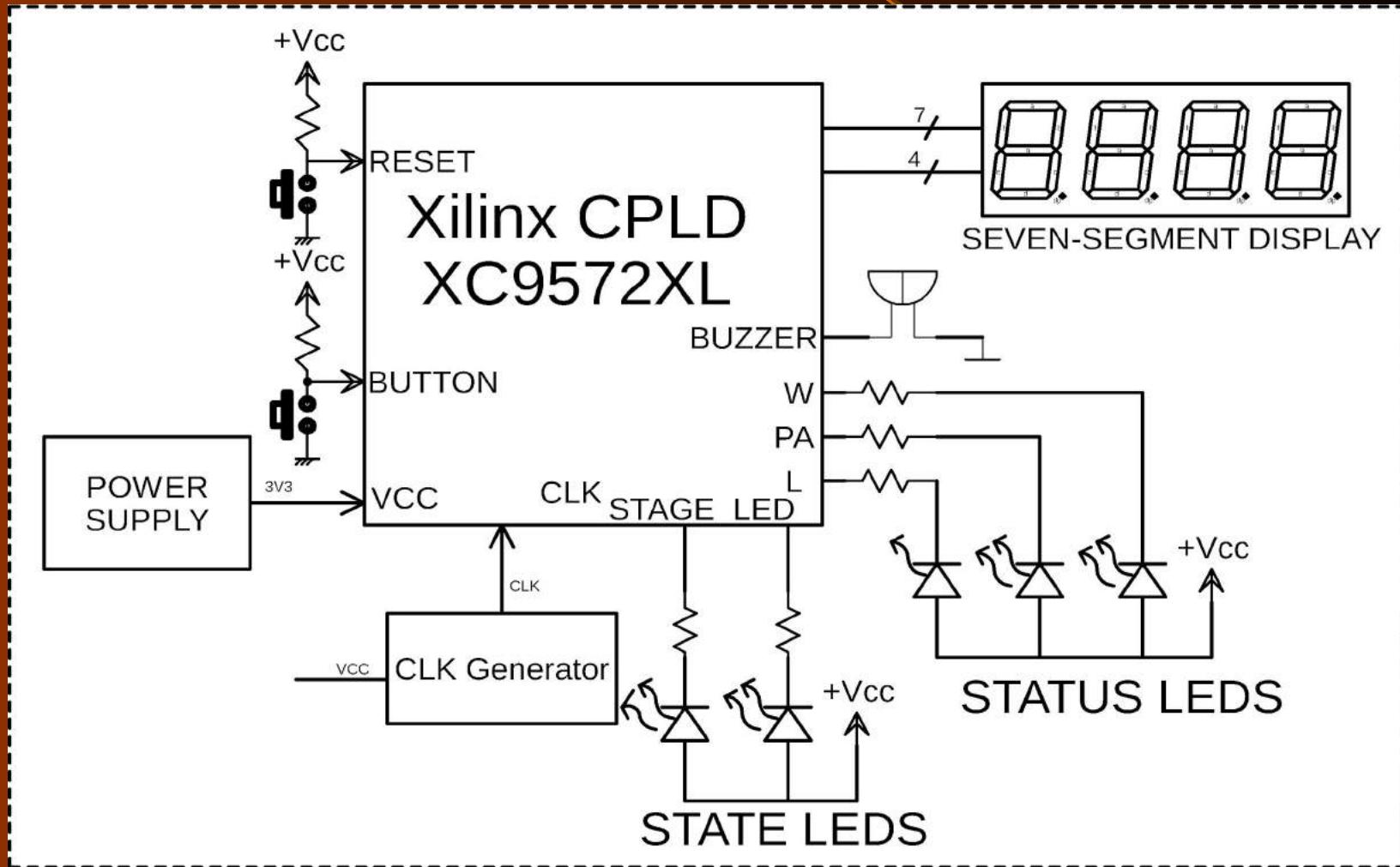
WIN PA Lose
stage2 stage1

→ Active Low

display_enable → Active High

	stage2	stage1	display_enable	WIN	PA	Lose	LE
S0	X	X	X	X	0	X	0
S1	1	0	1	1	1	1	0
S2	1	0	1	1	1	1	1
S3	X	X	1	0	1	1	0
S4	X	X	1	1	1	0	0
S5	0	1	1	1	0	1	0
S6	0	1	1	1	1	1	0
S7	0	1	1	1	1	1	1
S8	X	X	0	X	X	X	0

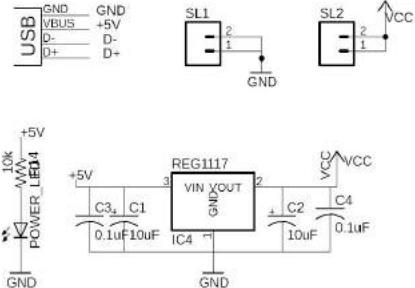
Top Level Block Diagram



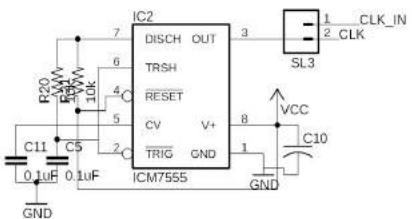
Schematic

CPLD BASED DICE GAME

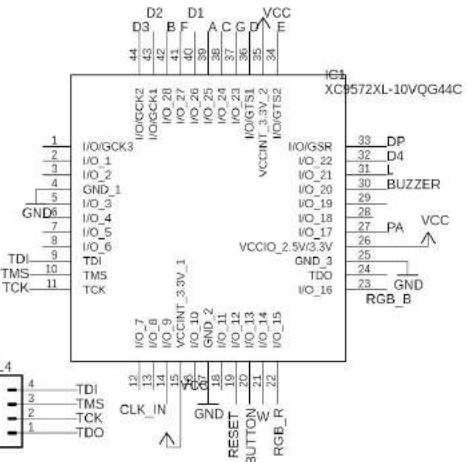
POWER SUPPLY



CLK GENERATOR

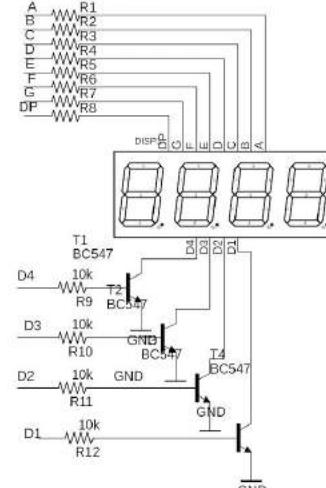


CPLD PINOUT

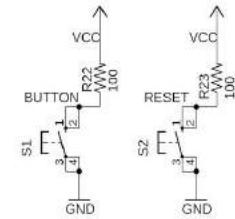


XC9572XL-VQ4410C

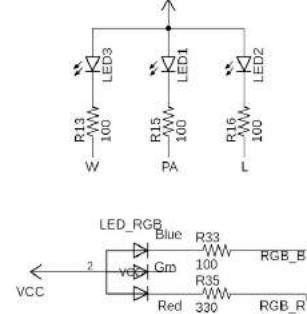
SEVEN SEGMENT DISPLAY



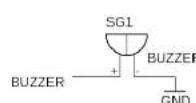
INPUT BUTTONS



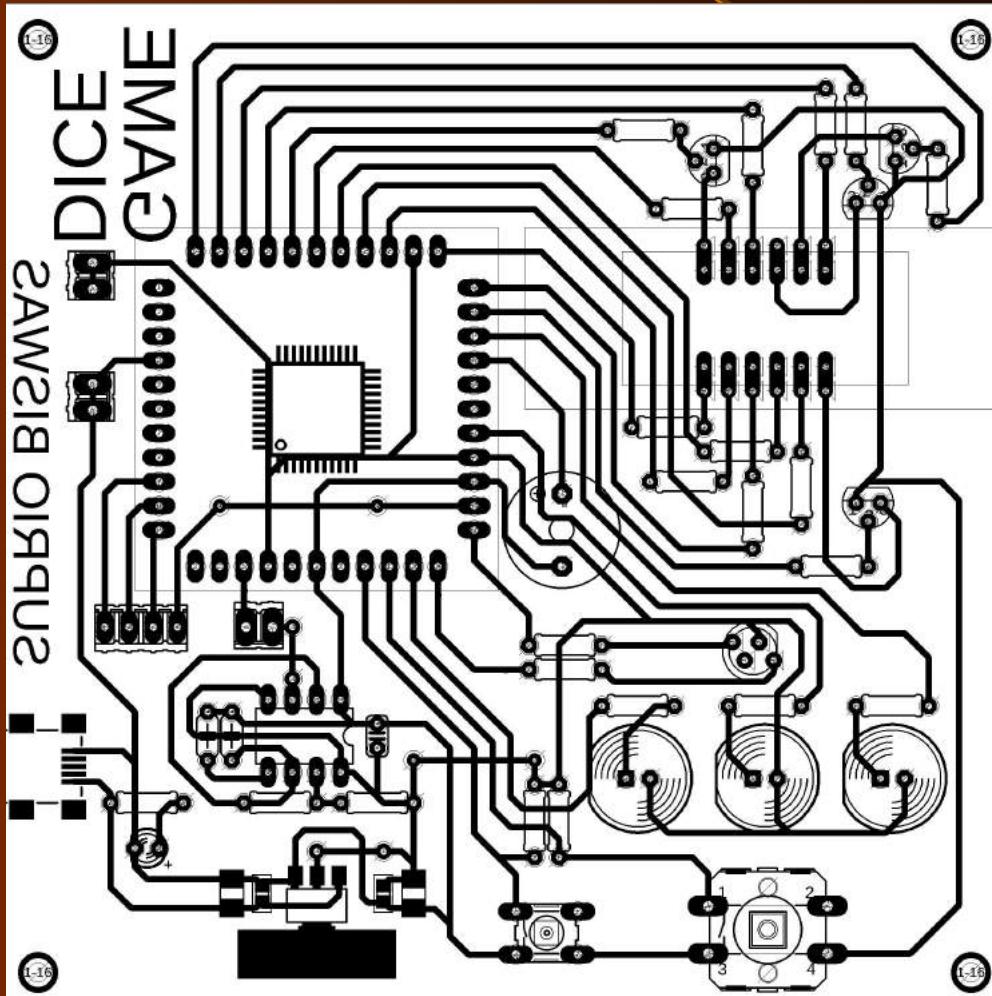
OUTPUT LEDS



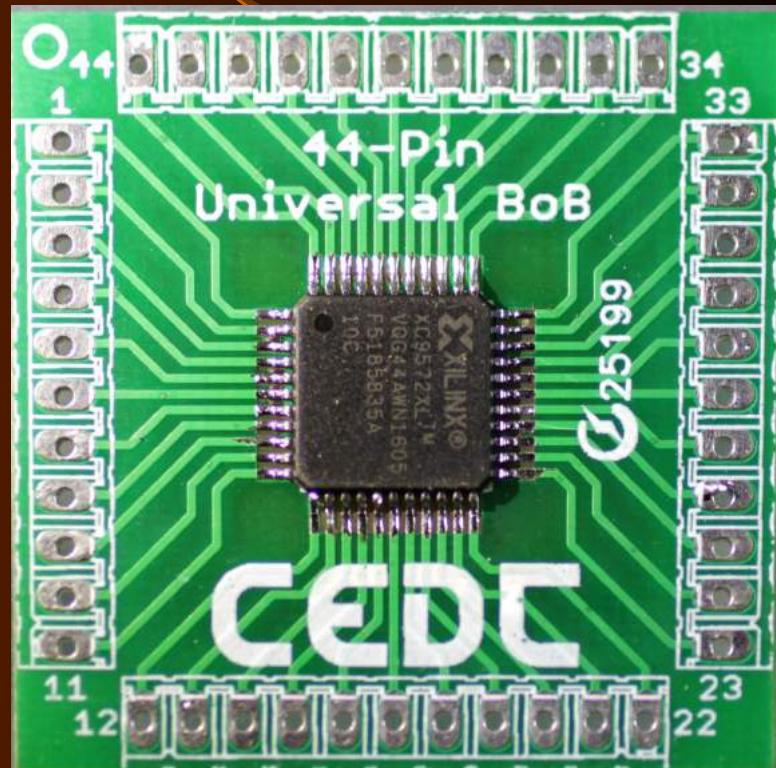
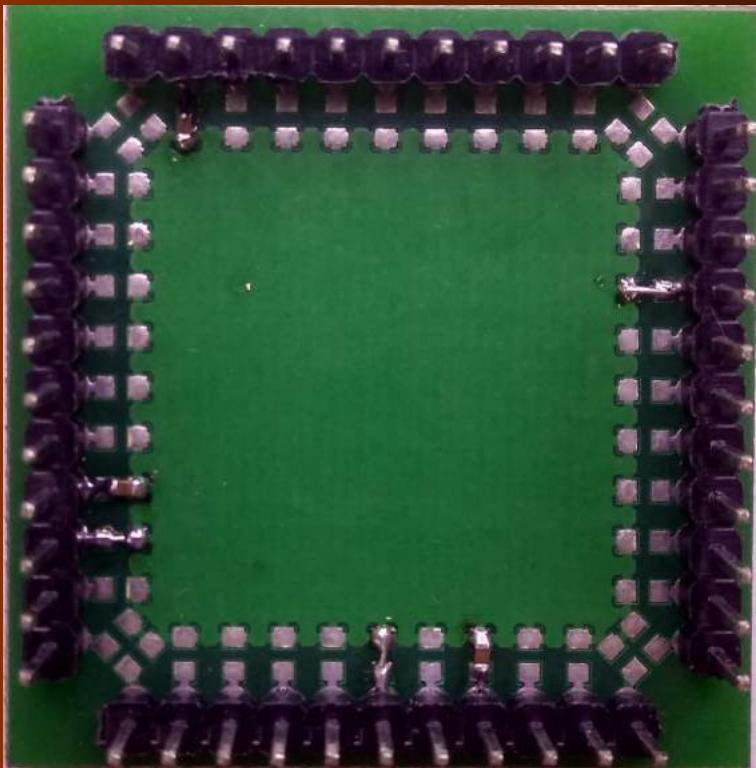
BUZZER



Board Layout



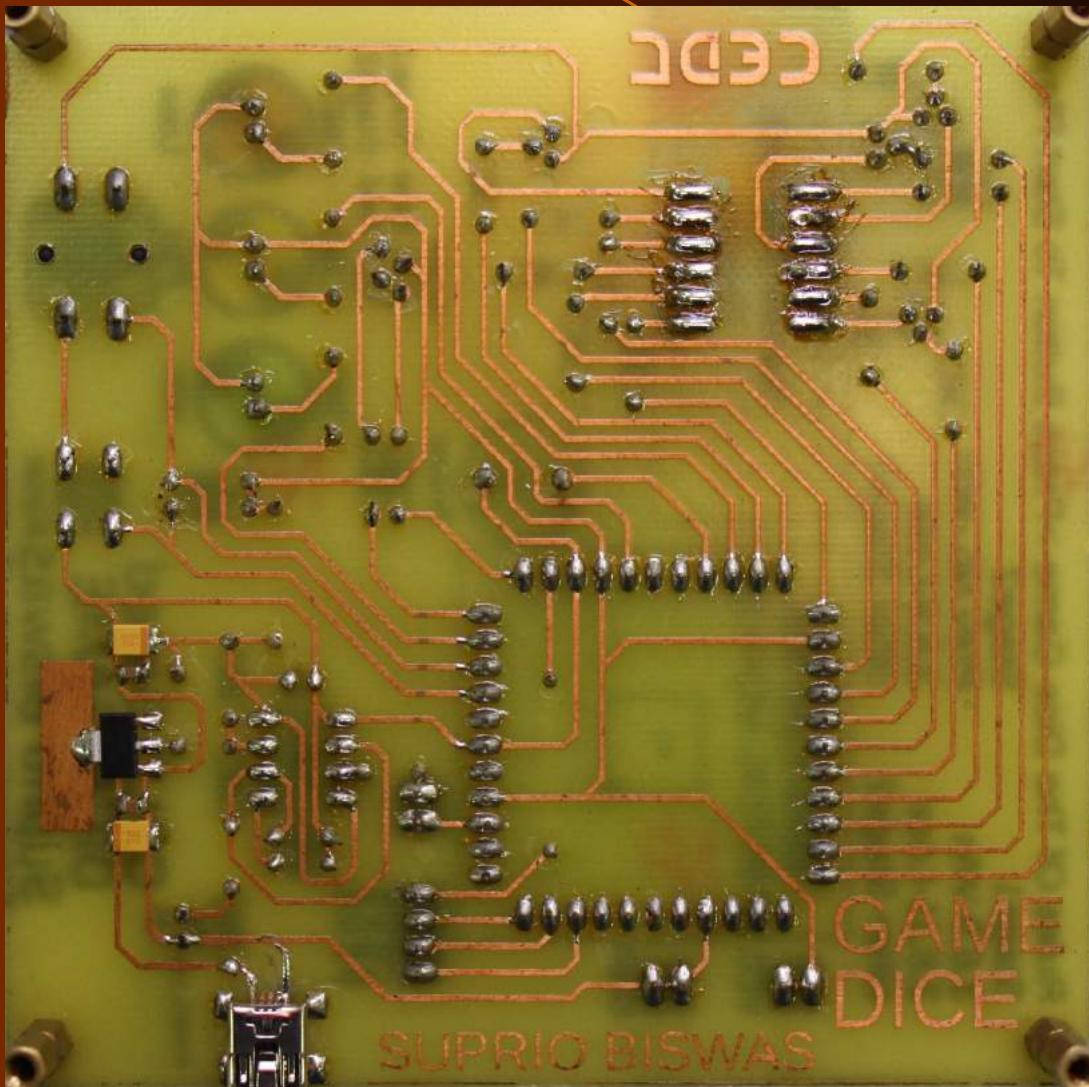
Break Out Board



Top Layer Of PCB



Bottom Layer Of PCB





Thank you!

Introduction to Embedded System Design

Recap of Coverage and Project Demonstration from Concept to Final

Dhananjay V. Gadre

Associate Professor

ECE Division

Netaji Subhas University of
Technology, New Delhi

Badri Subudhi

Assistant Professor

Electrical Engineering Department

Indian Institute of Technology,
Jammu

Recap of Course Coverage

- Embedded Systems Characteristics
- Project Demonstrations
- Implementing the Embedded Computer
- Microcontroller as the Embedded Computer
- Single Purpose Computers
- Salient Features of Modern Microcontrollers
- Choosing the Right Microcontroller
- Elements of the Microcontroller Ecosystem
- Six-Box Model for an Embedded System
- Details of Input, Output, Controller, Power-supply, Communication and Host/Storage Box

Recap of Course Coverage

- Idea about Electronic Glue
- Design of Power Supply
- MSP430 Architecture
- CCS and GIT
- Embedded C Programming
- MSP430 Peripherals - Digital I/O
- MSP430 Clock, Reset and Low Power Modes
- Physical Interfacing - Driving Outputs
- Physical Interfacing - Reading Inputs
- Connecting Seven Segment Displays and LCD
- MSP430 Interrupts

Recap of Course Coverage

- MSP430 Timer Compare, Capture
- PWM
- MSP430 ADC
- ADC Interfacing Issues. Reading Sensors
- Generating Random numbers,
- DAC
- MSP430 Serial Communication
- Ninja Level Programming
- Project Planning
- Circuit Prototyping
- And This Lecture!

Implementing an MSP430 Project

- Objective
- Visualization
- Planning
- Module Testing
- Schematic and PCB layout
- Circuit Prototyping
- Implementation
- Documentation

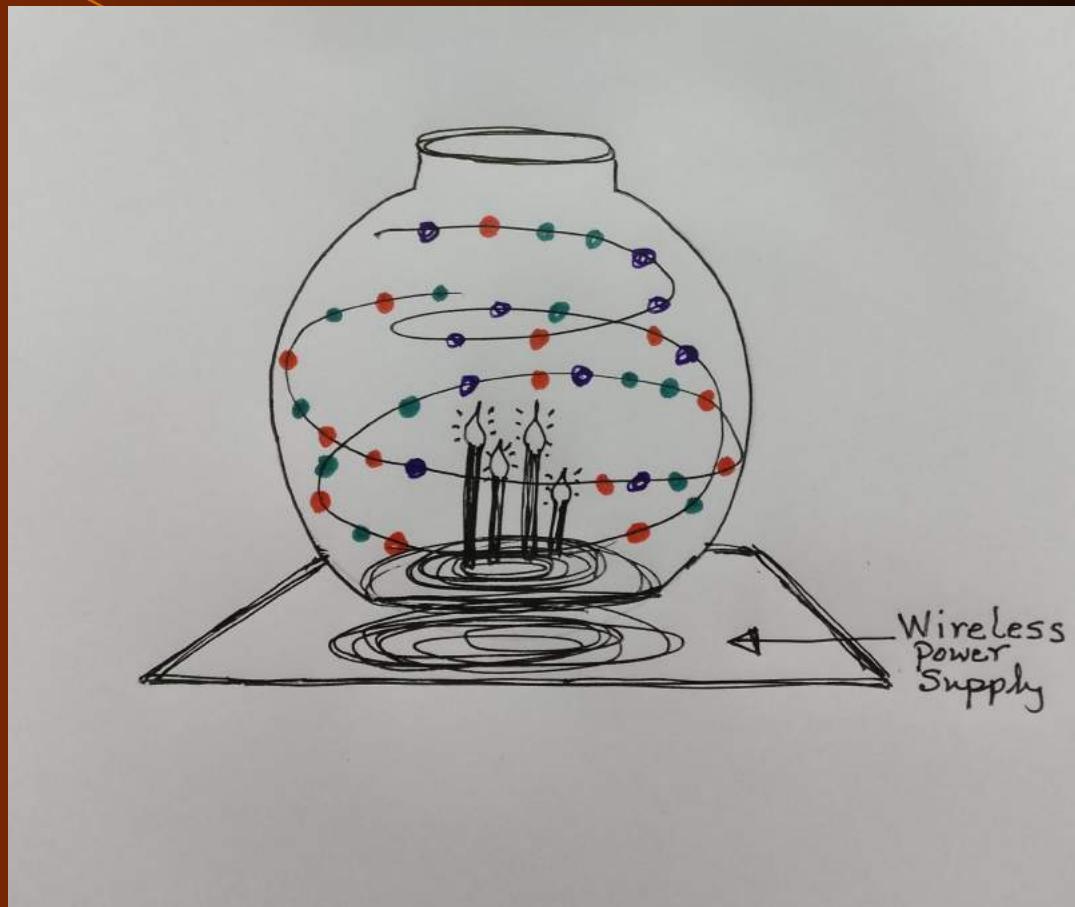
Wirelessly Powered Art Installation

- Objective



Wirelessly Powered Art Installation

- Visualization



Wirelessly Powered Art Installation

- Planning
 1. Wireless Power Supply
 2. Microcontroller Circuit

Wireless Power Transfer

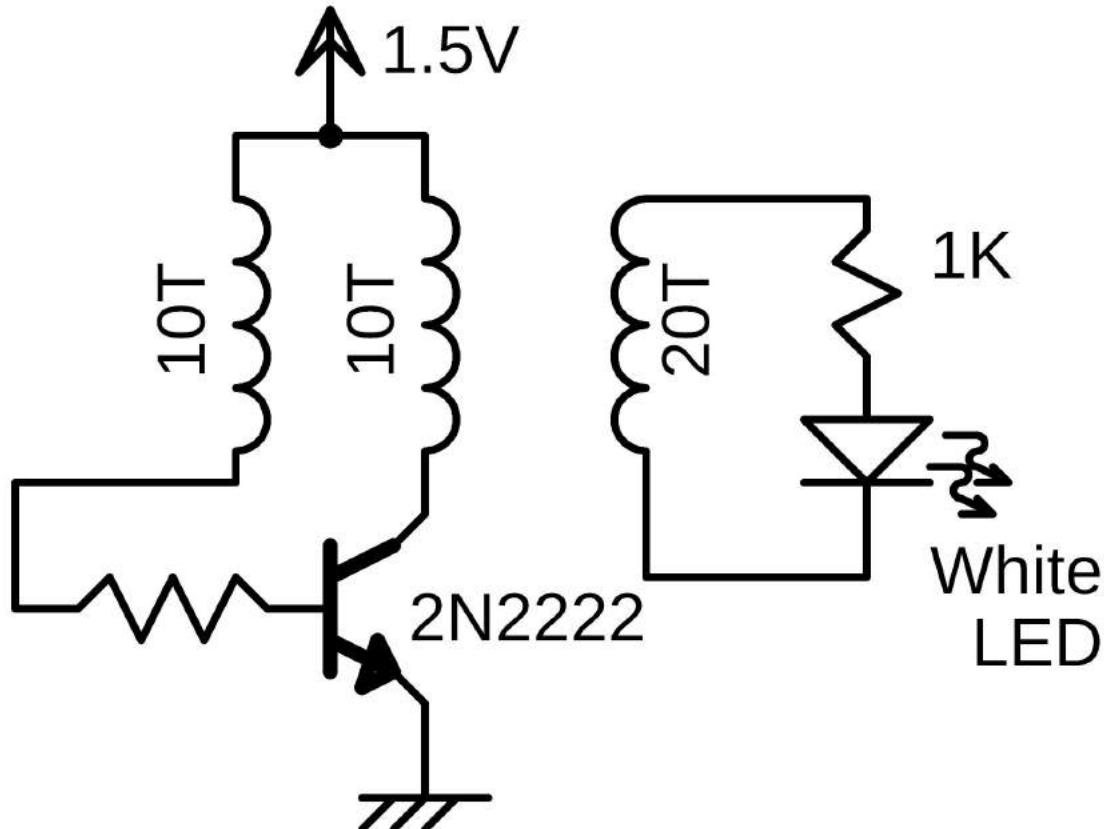
Transmission of energy without wires.

1. Time varying EM field which transmits power over space
2. Receiver to extract power.

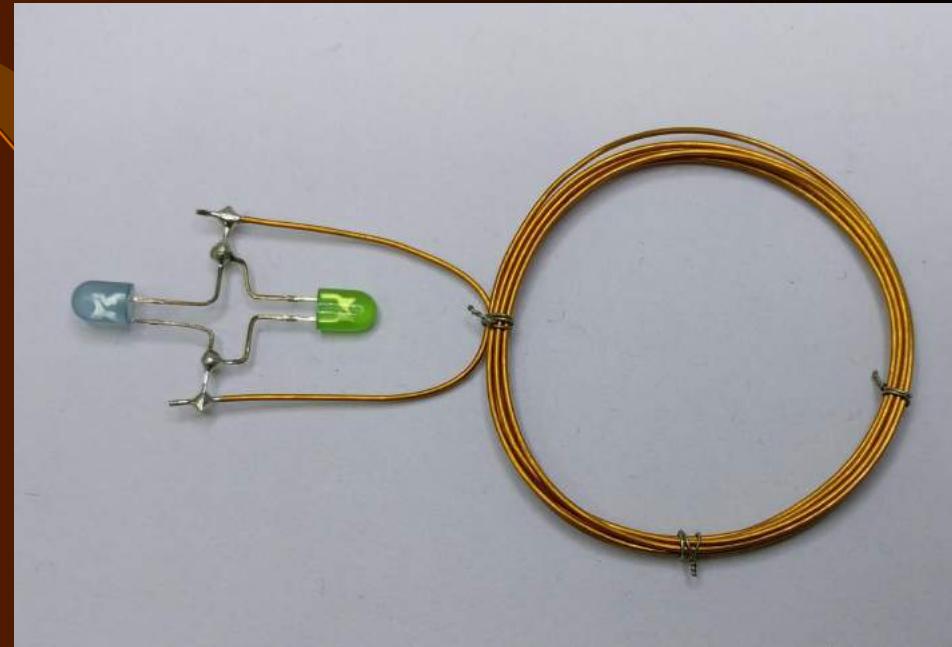
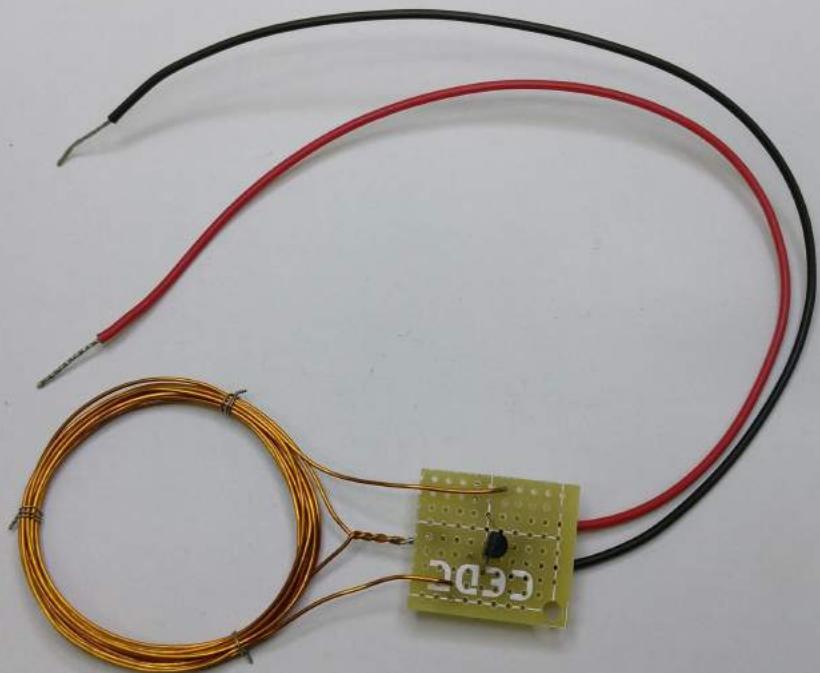
Wireless Power Transfer

1. Near Field - Non-radiative.
 - Magnetic fields using Inductive Coupling using coils
 - Electric Fields using Capacitive Coupling using electrodes
2. Far Field - Radiative method.

Wireless Power Supply



Wireless Power Supply



Wireless Power Supply

Higher Voltage for transmitter

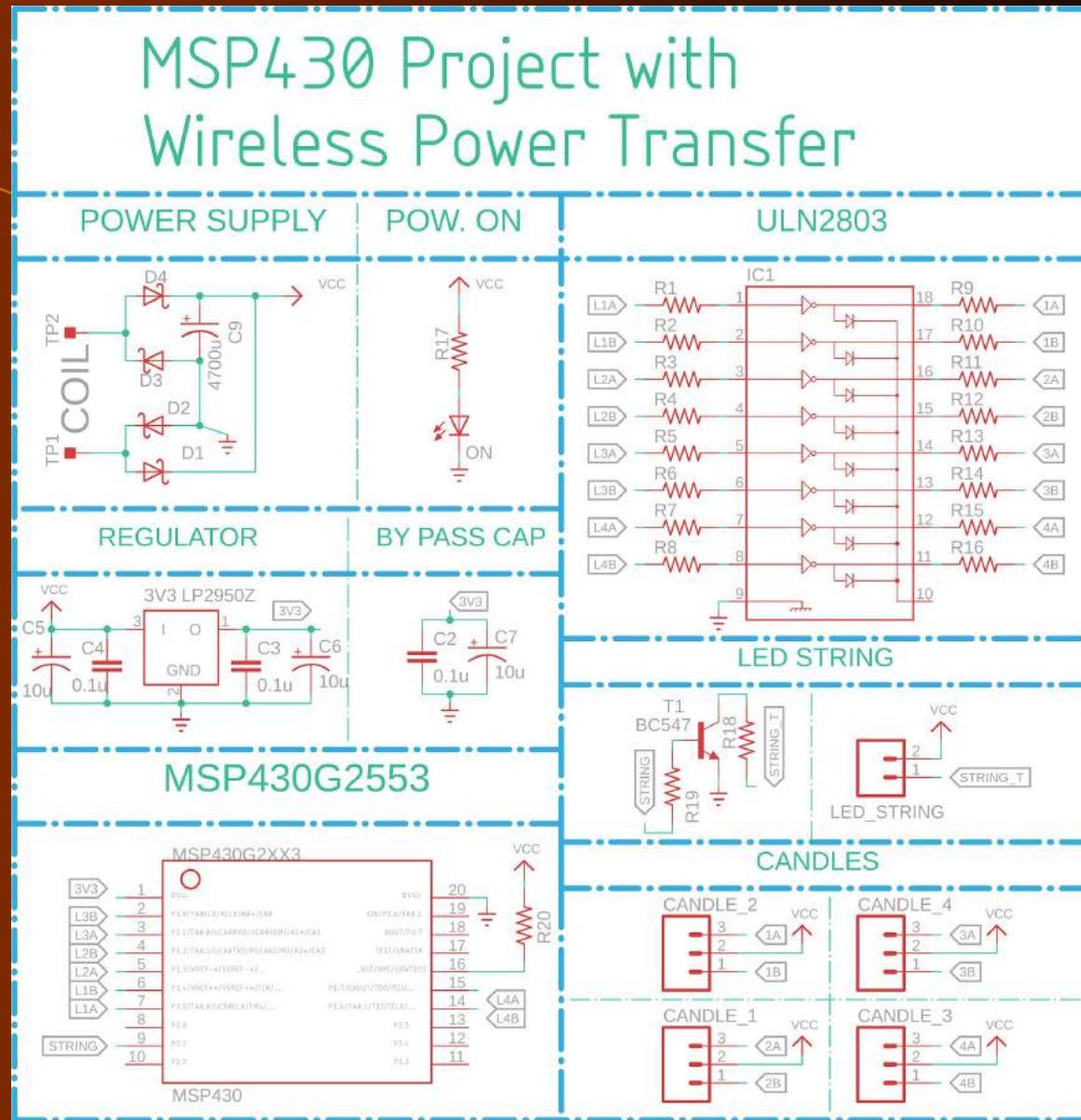
Higher power dissipation transistor

At 6V primary, $V_{out} = 4.5V$, 100mA

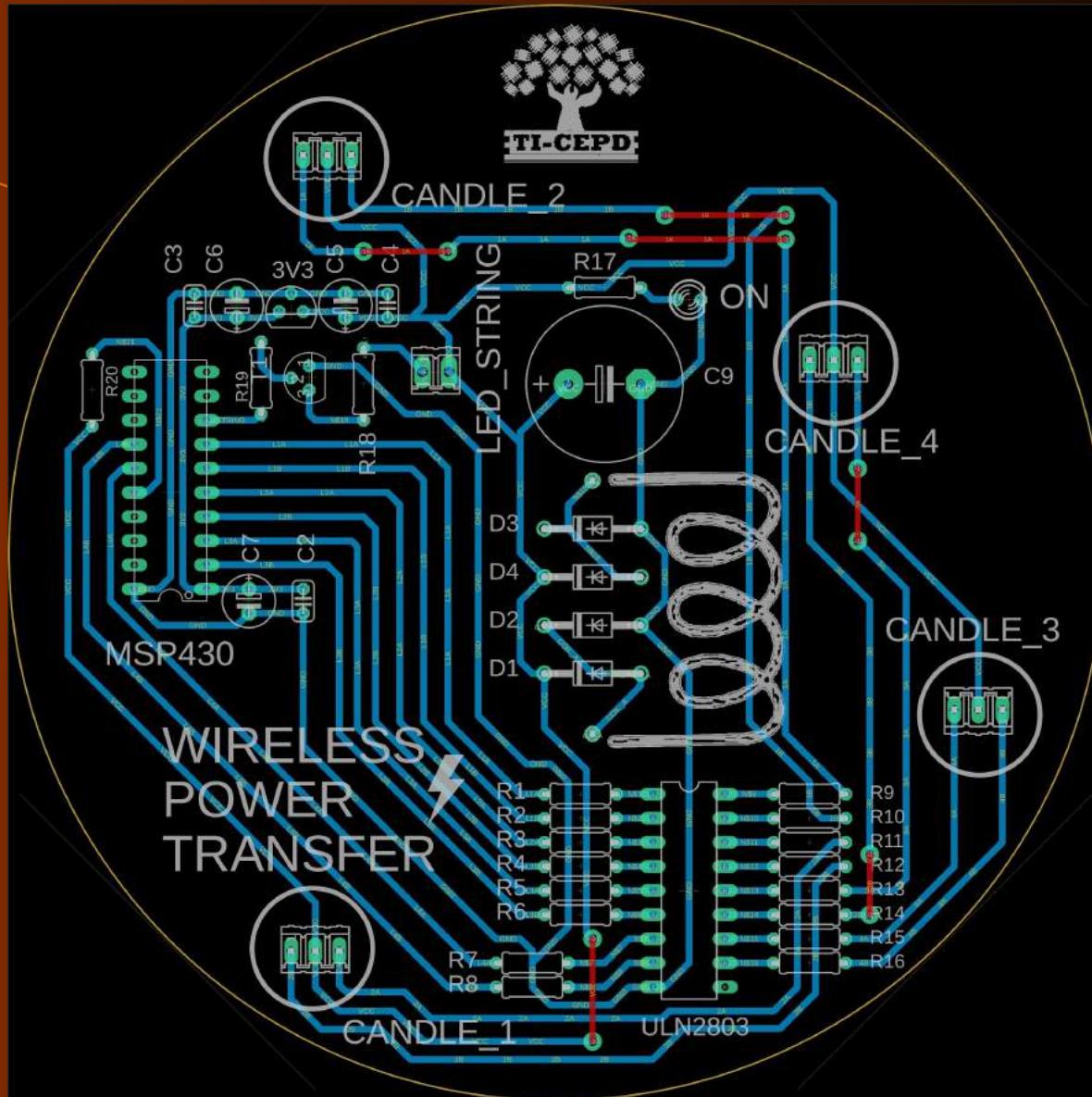
Microcontroller Circuit

1. Few white LEDs with flickering effect
2. Multicolored LED string with lighting patterns

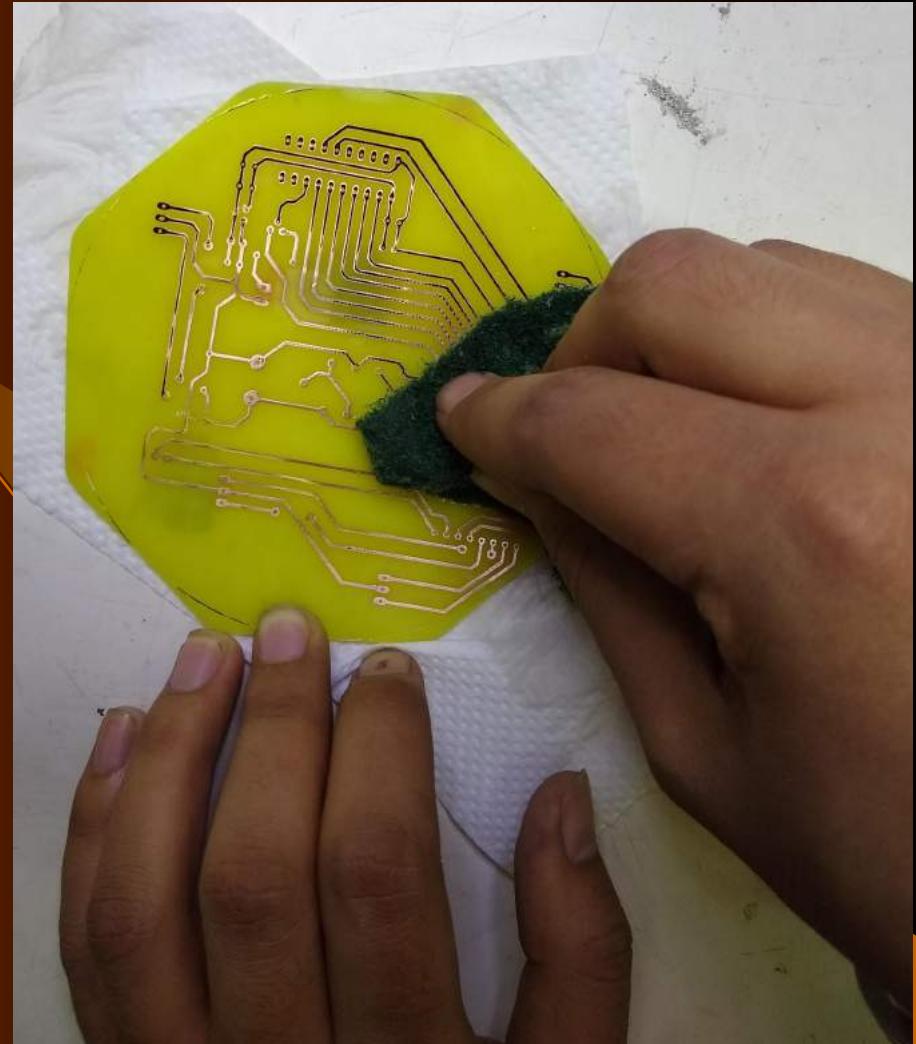
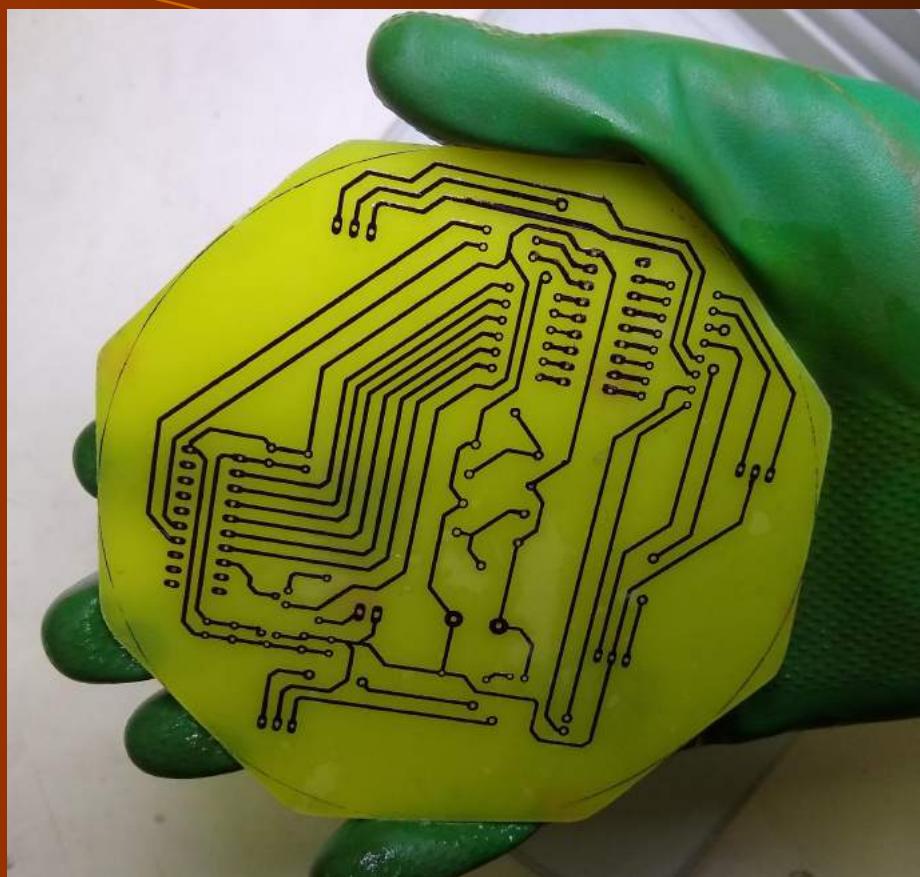
Schematic and PCB layout



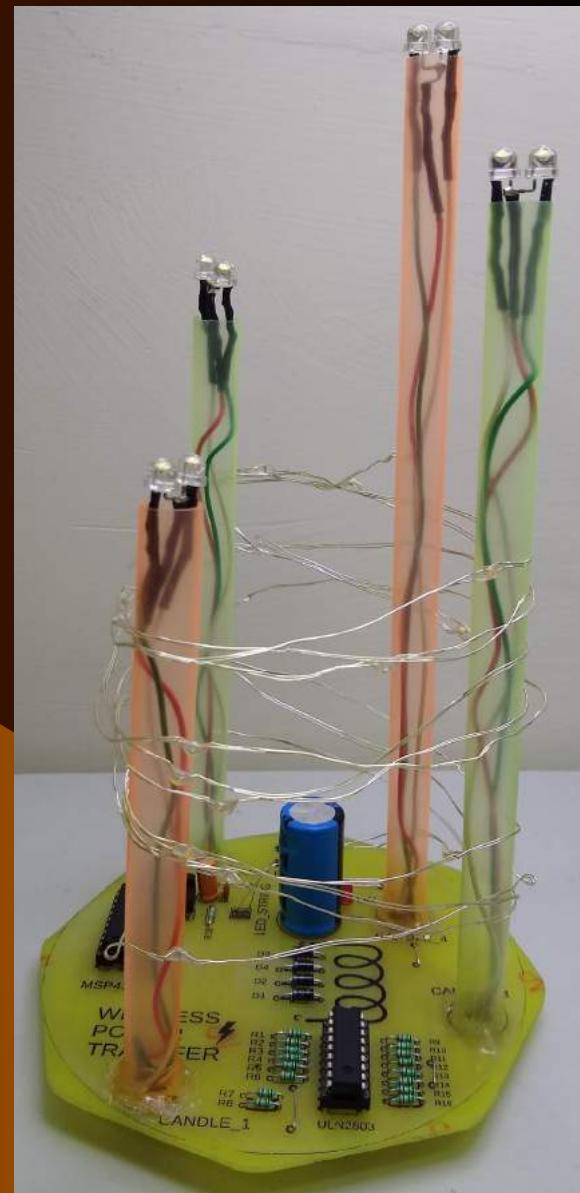
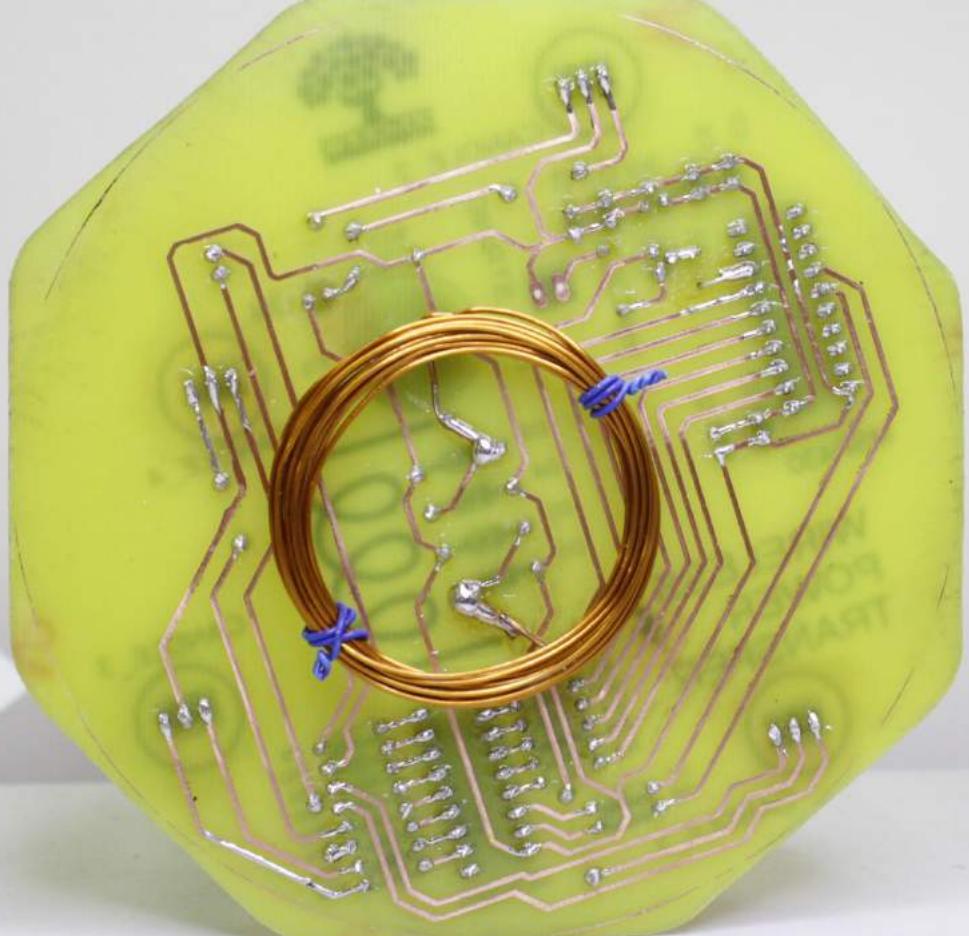
Schematic and PCB layout



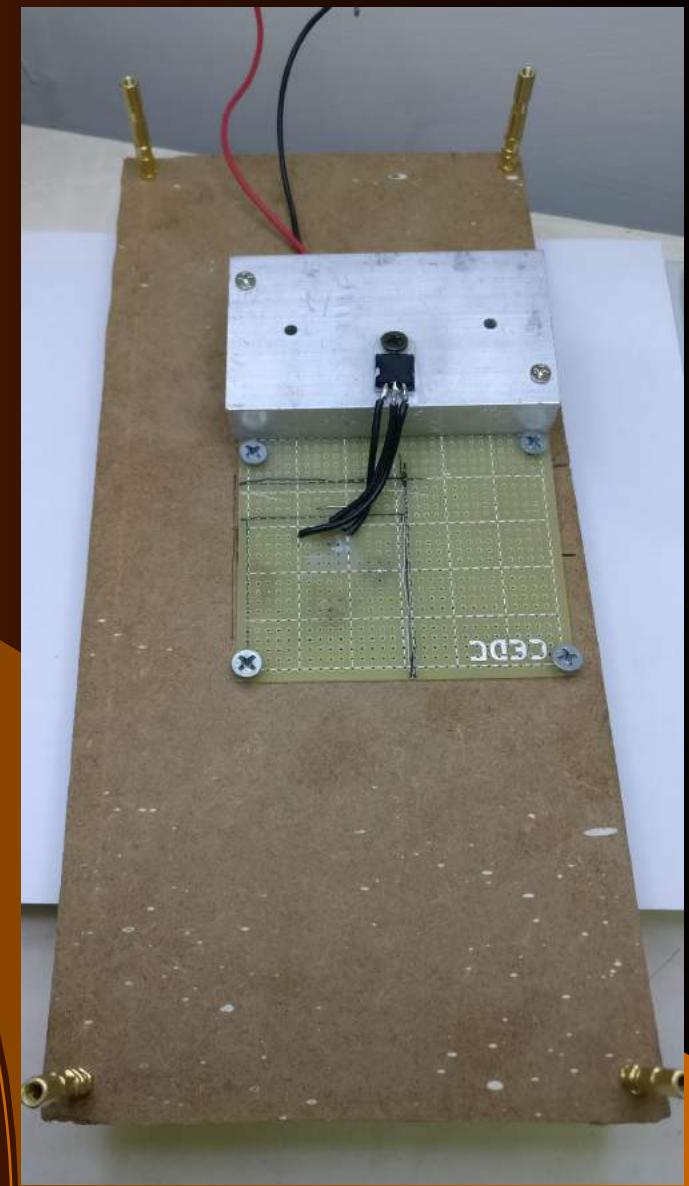
Fabrication



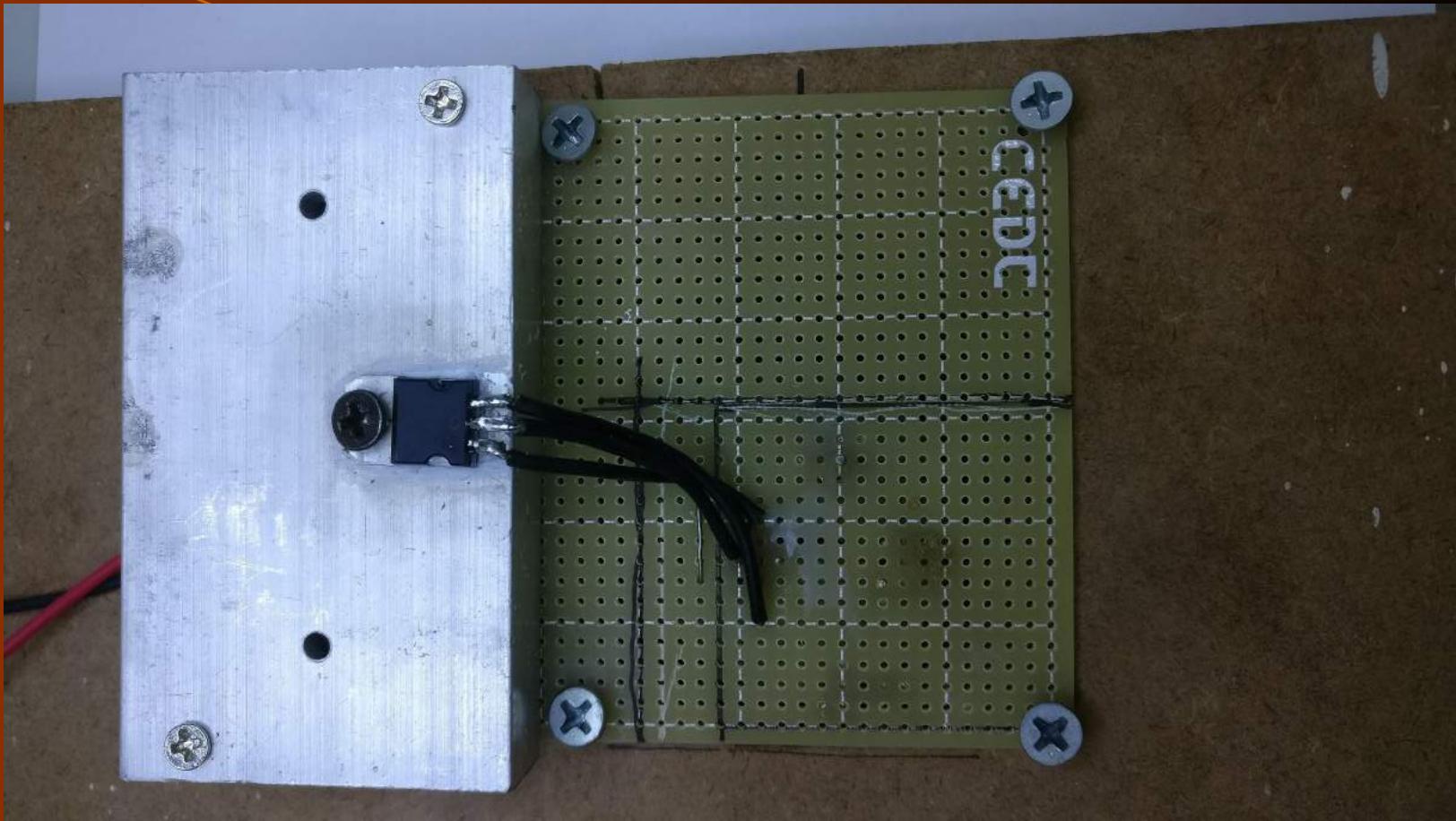
Soldering



Power Supply Transmitter Base



Power Supply Transmitter Base



Assembled Version



Embedded Code

```
1 #include <msp430.h>
2 #include <stdint.h>
3
4 #define LED      BIT1          // LED Strip -> P2.1
5 #define UP       1             //Increasing Brightness
6 #define DOWN    0             //Decreasing Brightness
7 #define BLINK   2             // For blink of leds
8
9
10 /**
11 * @brief
12 * These settings are wrt enabling TIMER1 on Lunchbox
13 */
14 void register_settings_for_TIMER1()
15 {
16     P2DIR |= LED;           // LED -> Output
17     P2SEL |= LED;           // LED -> Select Timer Output
18
19     TA1CCR0 = 255;          // Set Timer0 PWM Period
20     TA1CCTL1 = OUTMOD_7;    // Set TA1.1 Waveform Mode - Clear on Compare, Set on Overflow
21     TA1CCR1 = 0;            // Set TA1.1 PWM duty cycle
22     TA1CCTL0 = CCIE;        // CCR0 Enable Interrupt
23     TA1CTL = TASSEL_2 + MC_1; // Timer Clock -> SMCLK, Mode -> Up
24 }
25
26 volatile int dir = UP;
27 volatile int count = 0; //for delay between change of dutycycle
28 volatile int count2 = 0; //for delay between blinking
29 volatile int num_blinks = 0; //for number of blinks
30
```

```
30
31 /*brief entry point for TIMER1_interrupt vector */
32 #pragma vector = TIMER1_A0_VECTOR
33 __interrupt void Timer_A(void){
34     count++;
35     if(count>100){                                //enter if after 100 interrupts
36         switch(dir){
37             case UP:
38                 TA1CCR1 = TA1CCR1+1;           //increment CCR1
39                 if(TA1CCR1>TA1CCR0){
40                     dir = BLINK;            //change dir to BLINK
41                     count2=0;              //initialize count2
42                     num_blinks=10;        //initialize num_blinks
43                 }
44                 break;
45             case BLINK:
46                 count2++;
47                 if(count2>10){           //do not change state of leds till count2 becomes 10
48                     num_blinks--;          //decrement no. of blinks
49                     if(num_blinks>0){
50                         if(TA1CCR1==0)
51                             TA1CCR1 = 255;    //full brightness
52                         else{
53                             TA1CCR1=0;        //zero brightness
54                         }
55                     }else{
56                         TA1CCR1=255;        //full brightness
57                         dir = DOWN;       //change dir to decreasing brightness
58                     }
59                     count2 = 0;
60                 }
61                 break;
62             case DOWN:
63                 TA1CCR1 = TA1CCR1 - 1;    //decrement CCR1
64                 if(TA1CCR1==0){
65                     dir = UP;            //change direction to increasing
66                 }
67                 break;
68         }
69         count=0;
70     }
71 }
```

```
71 }
72 // @brief entry point for the code/
73 void main(void) {
74     P1DIR |= 0xFF;
75     unsigned int i;
76     uint32_t lfsr = 0x81283723;      //seed
77     uint32_t bit = 0x00000000;
78
79     WDTCTL = WDTPW | WDTHOLD;          // Stop watchdog timer
80     register_settings_for_TIMER1();
81     __bis_SR_register(GIE);           // Enable CPU Interrupt
82     while(1){
83         P1OUT = lfsr & 0xFF;
84         bit = ((lfsr >> 31) ^ (lfsr >> 21) ^ (lfsr >> 1) ^ (lfsr >> 0));
85         lfsr = (lfsr >> 1) | (bit << 31);                      //feedback
86         for(i=2500; i>0; i--);                                // delay
87     }
88 }
```

Wirelessly Powered Art Installation

- Documentation



Thank you!