2.996/6.971 Biomedical Devices Design Laboratory

Lecture 6: Microprocessors II

Instructor: Dr. Hong Ma

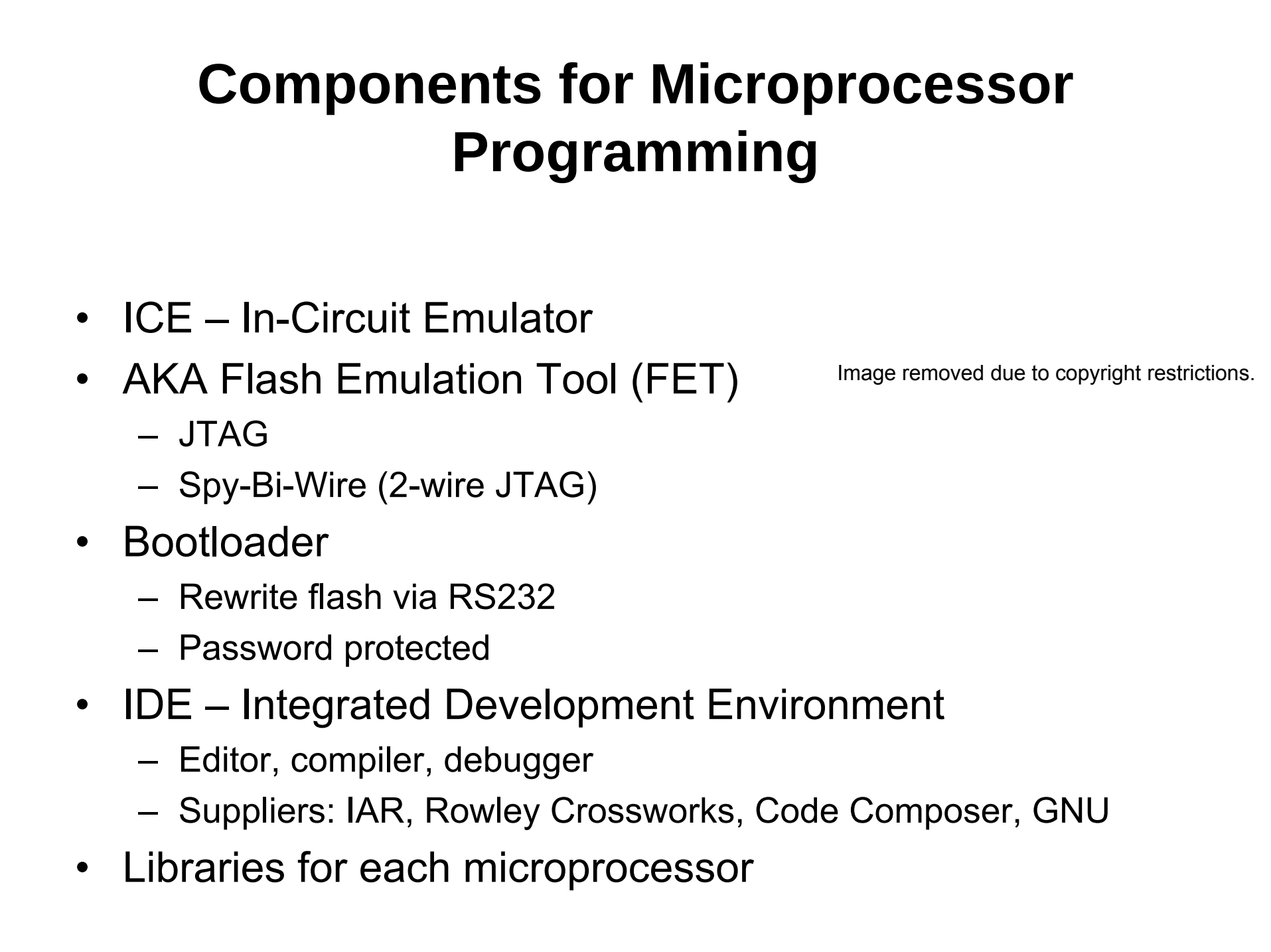
Oct. 1, 2007

Structure of MSP430 Program

1. Declarations
2. `main()`
    1. Watch-dog timer servicing
    2. Setup clocking module
    3. Setup peripheral modules  **Key**
    4. Enable interrupts
    5. Infinite loop
3. Subroutines
4. Interrupt Service Routines (ISR)

Components for Microprocessor Programming

- ICE – In-Circuit Emulator
- AKA Flash Emulation Tool (FET)
  - JTAG
  - Spy-Bi-Wire (2-wire JTAG)



- Bootloader
  - Rewrite flash via RS232
  - Password protected
- IDE – Integrated Development Environment
  - Editor, compiler, debugger
  - Suppliers: IAR, Rowley Crossworks, Code Composer, GNU
- Libraries for each microprocessor

Variable Types

Type	Size	Single-cycle instruction?
char	8-bits	Yes
int	16-bits	Yes
long	32-bits	No
float	32 or 64-bits	No

Number Representation

- One's Complement (standard binary)

1	1	1	1	1	1	1	1	=	256
0	1	1	1	1	1	1	1	=	127
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0

8-bit one's complement integers

- Two's Complement

sign bit									
0	1	1	1	1	1	1	1	=	127
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0
1	1	1	1	1	1	1	1	=	−1
1	1	1	1	1	1	1	0	=	−2
1	0	0	0	0	0	0	1	=	−127
1	0	0	0	0	0	0	0	=	−128

8-bit two's complement integers

```
//One's comp. definition
unsigned char var1
unsigned int var2
```

```
//Two's comp. definition
signed char var1
signed int var2
```

Always explicitly define signed / unsigned !!!

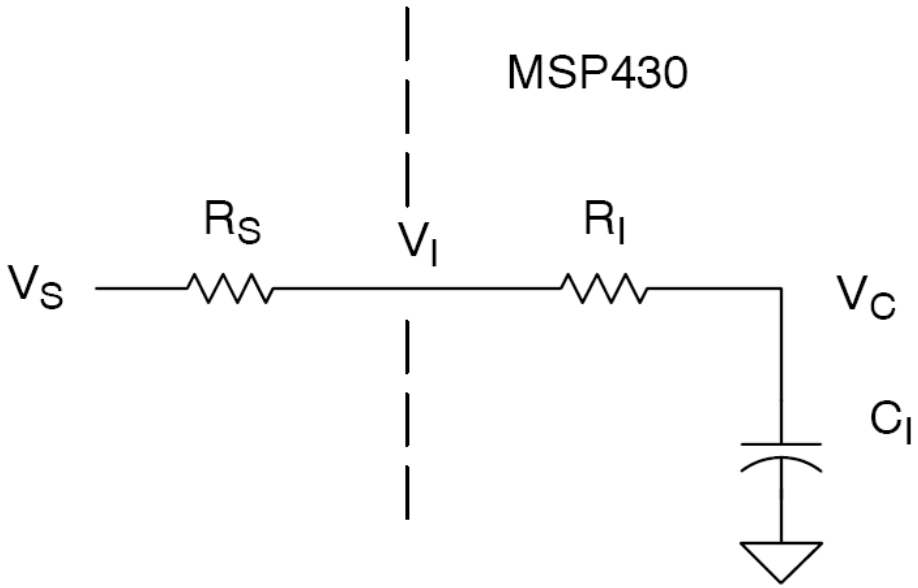
Global Variables

- It is often convenient to use global variables
- Global variables are not always updated between sub-routines due to compiler optimization
- Always declare ‘volatile’ for global variables

```
//Declarations
unsigned char var
volatile unsigned char gvar

Main()
{
gvar=1;
while(1);
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void UART_RX(void)
{
gvar=2;
}
```

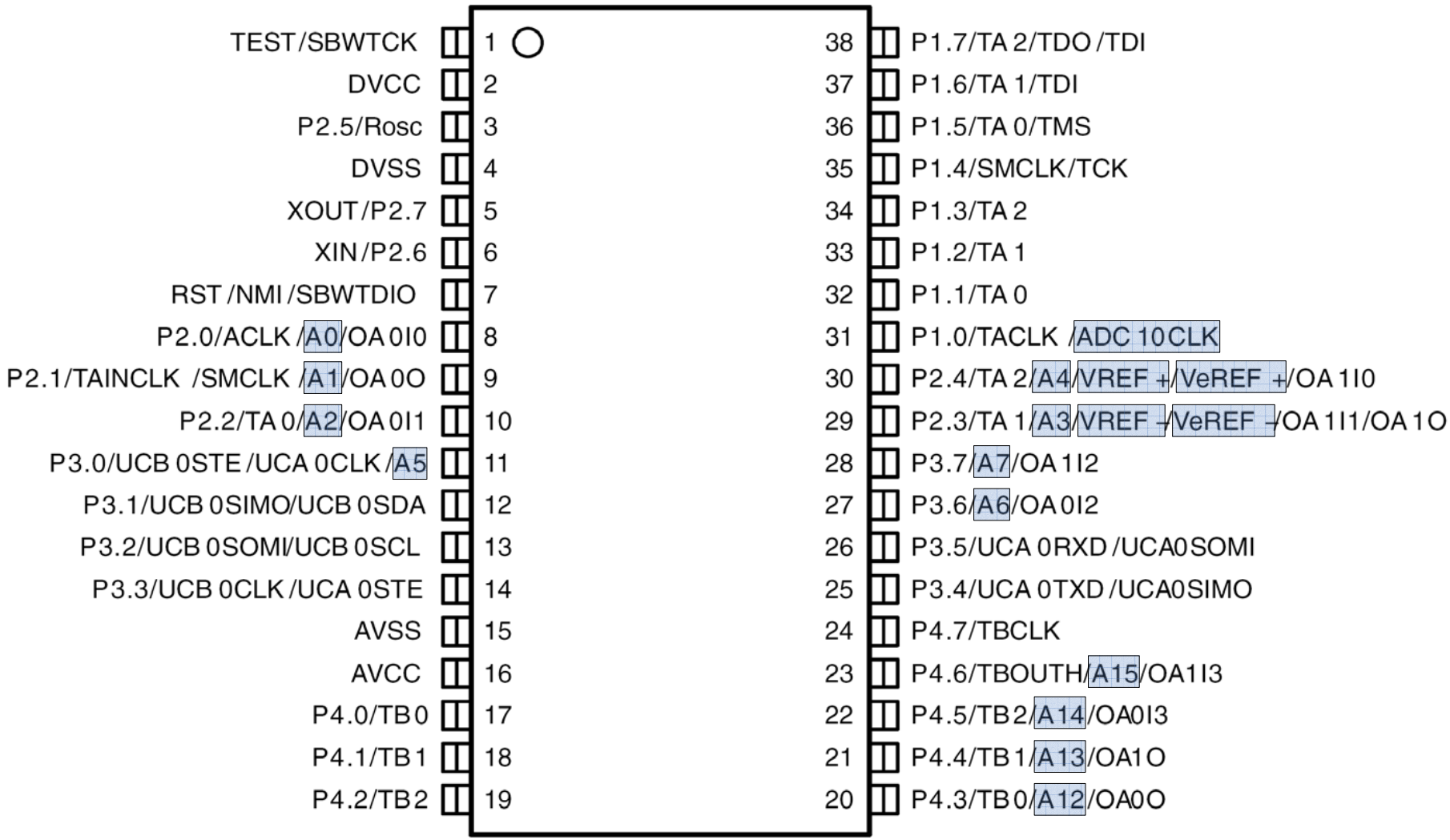
MSP430 10-bit ADC

- SAR architecture – “Snap-shot” type
- Up to 200ksps
- 12 multiplexed inputs
- Remember to check settling time:
  - C_I = 27pF
  - R_I = 2kΩ

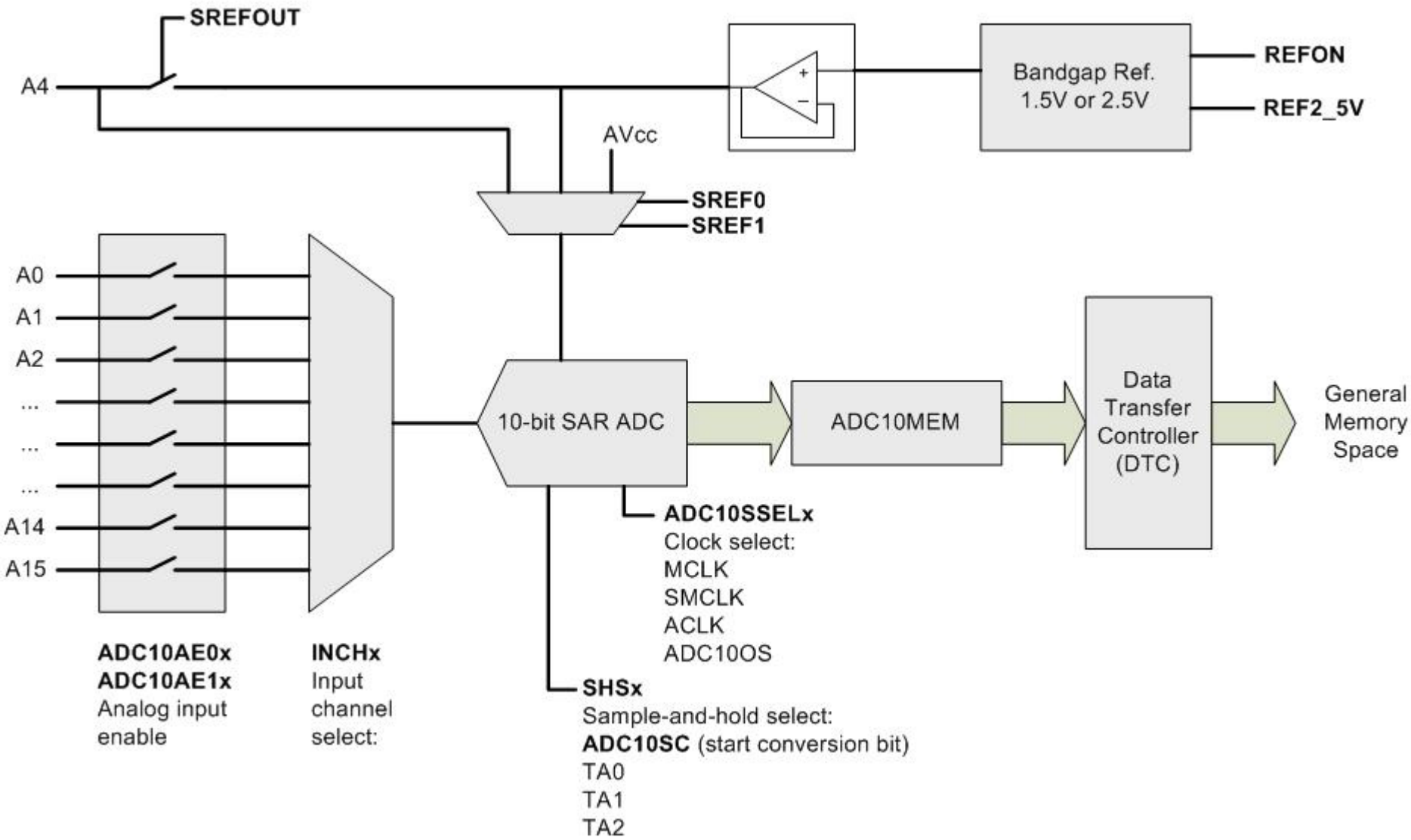


Pin-Outs for ADC

MSP430x22x4 device pinout, DA package



Simplified ADC Operating Schematic



Built-in Reference

- Internal bandgap reference
- 2.5V mode (2.35V – 2.65V)
- 1.5V mode (1.41V – 1.59V)
- Max load current ~ ±1mA
- Max load capacitance = 100pF
- Other reference sources:
  - AVcc
  - External

ADC Example Code

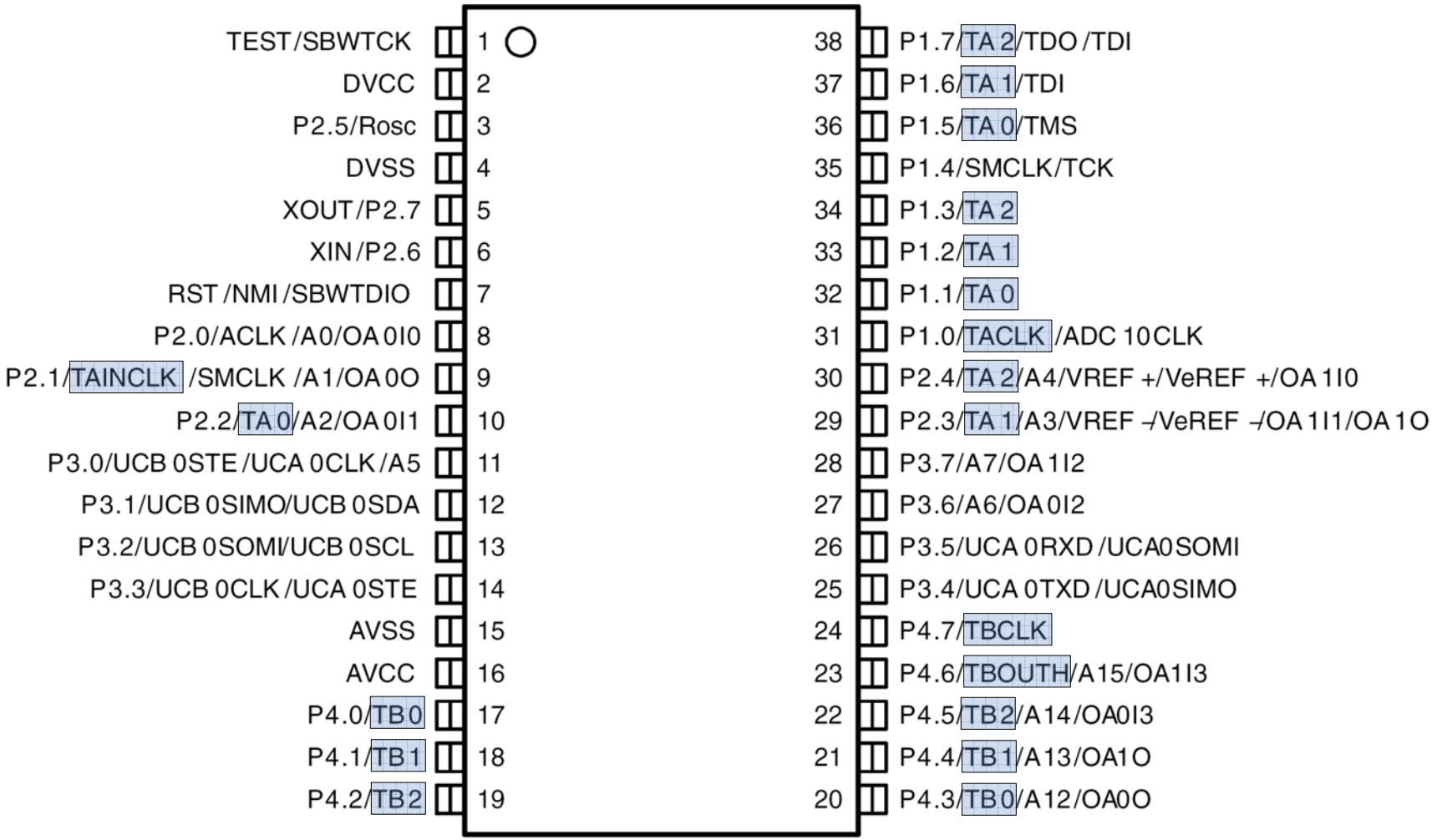
```
/*** set and output 2.5V reference, turn on ADC ***/
ADC10CTL0 = SREF0 | REFOUT | REF2_5V | REFON | ADC10ON;
/*** use ACLK, 16MHz, for conversion timing ***/
ADC10CTL1 = ADC10SSEL0;
/*** enable A1 as analog inputs ***/
ADC10AE1 = BIT1;
...
while(1) {
   /*** select channel_A0, use MCLK as ADC10CLK ***/
   ADC10CTL1 = INCH1 | ADC10SSEL1;
   /*** enable ADC and start conversion ***/
   ADC10CTL0 |= ENC | ADC10SC;
   /*** wait for conversion to be finished ***/
   while((ADC10CTL1 & ADC10BUSY) != 0);
       adc_result = ADC10MEM;
   ADC10CTL0 &= ~(ENC | ADC10SC);
}
```

Timers Functions

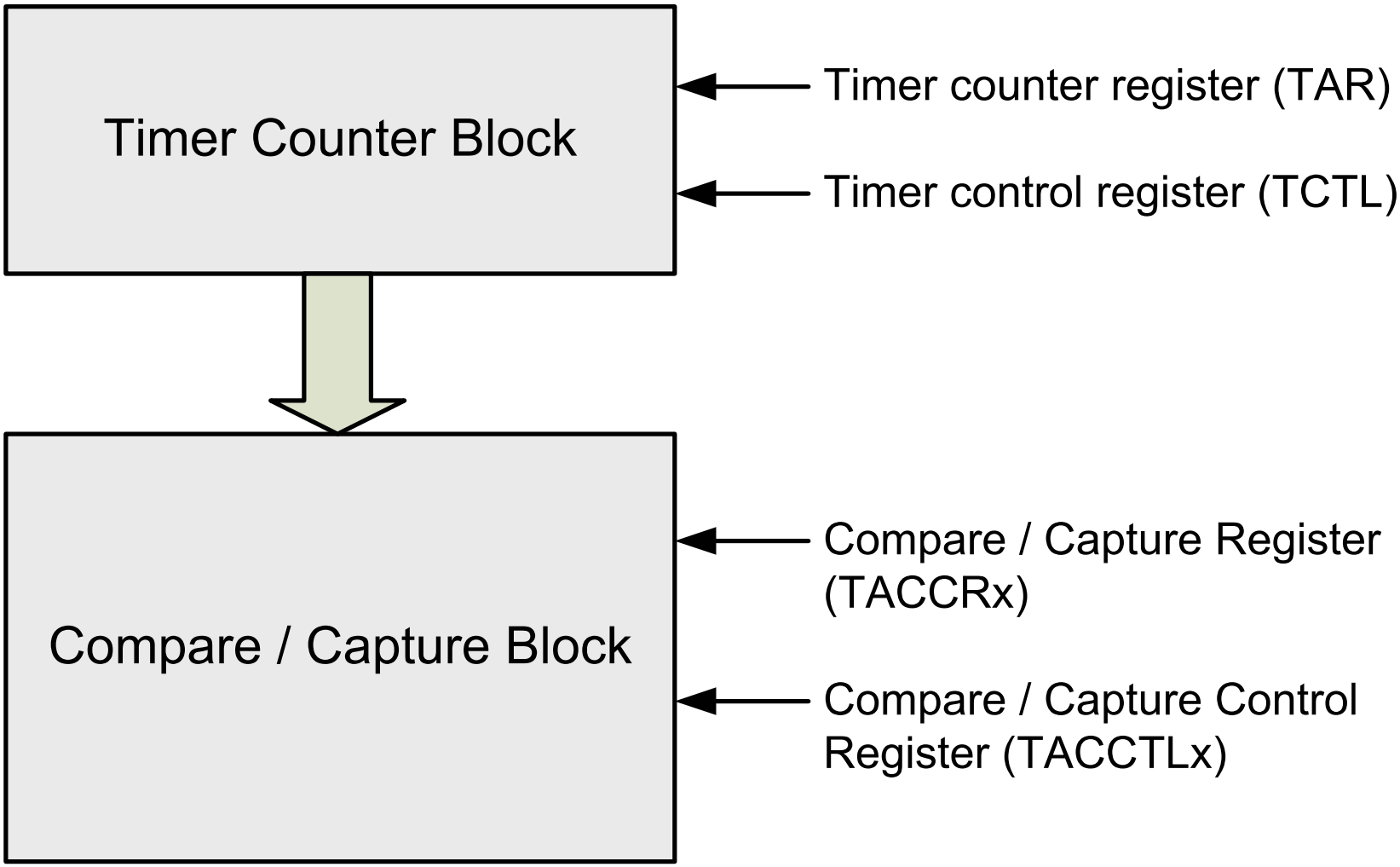
- Generate precise length pulses
  - Precise interval timing
  - PWM motor control
  - Simple DAC
- Measure pulse length
  - Sensing without ADC
  - Communications

Timer Related I/O

MSP430x22x4 device pinout, DA package



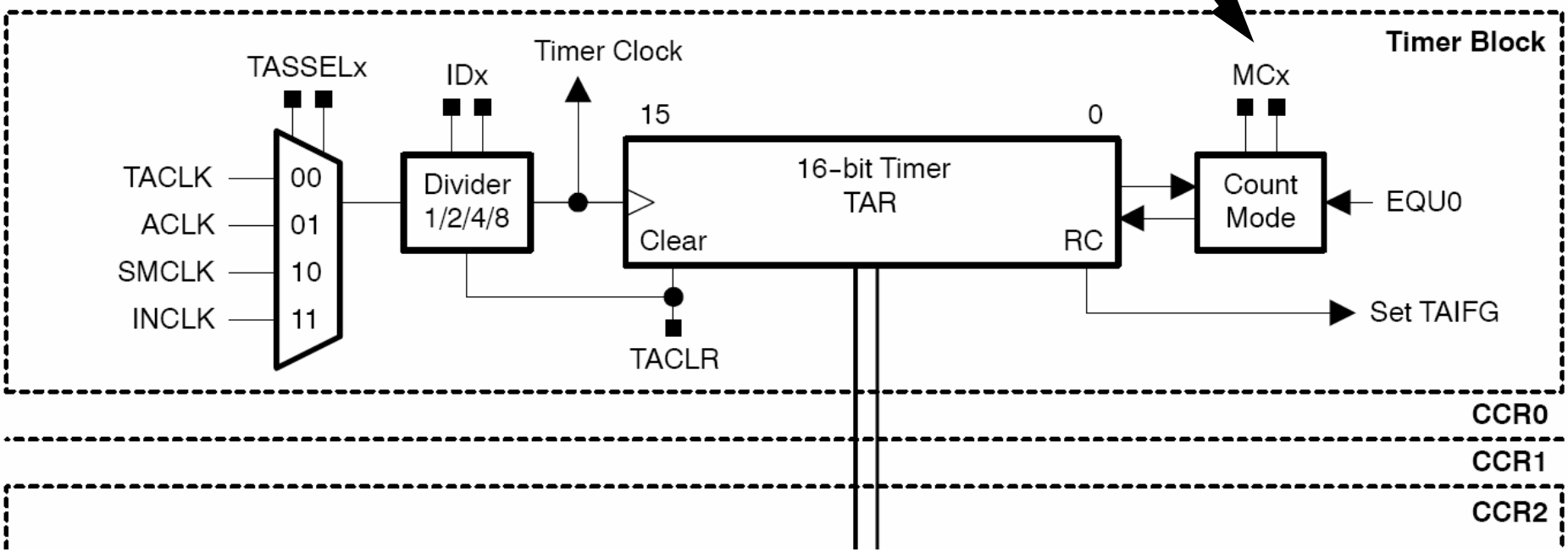
Timer Orientation



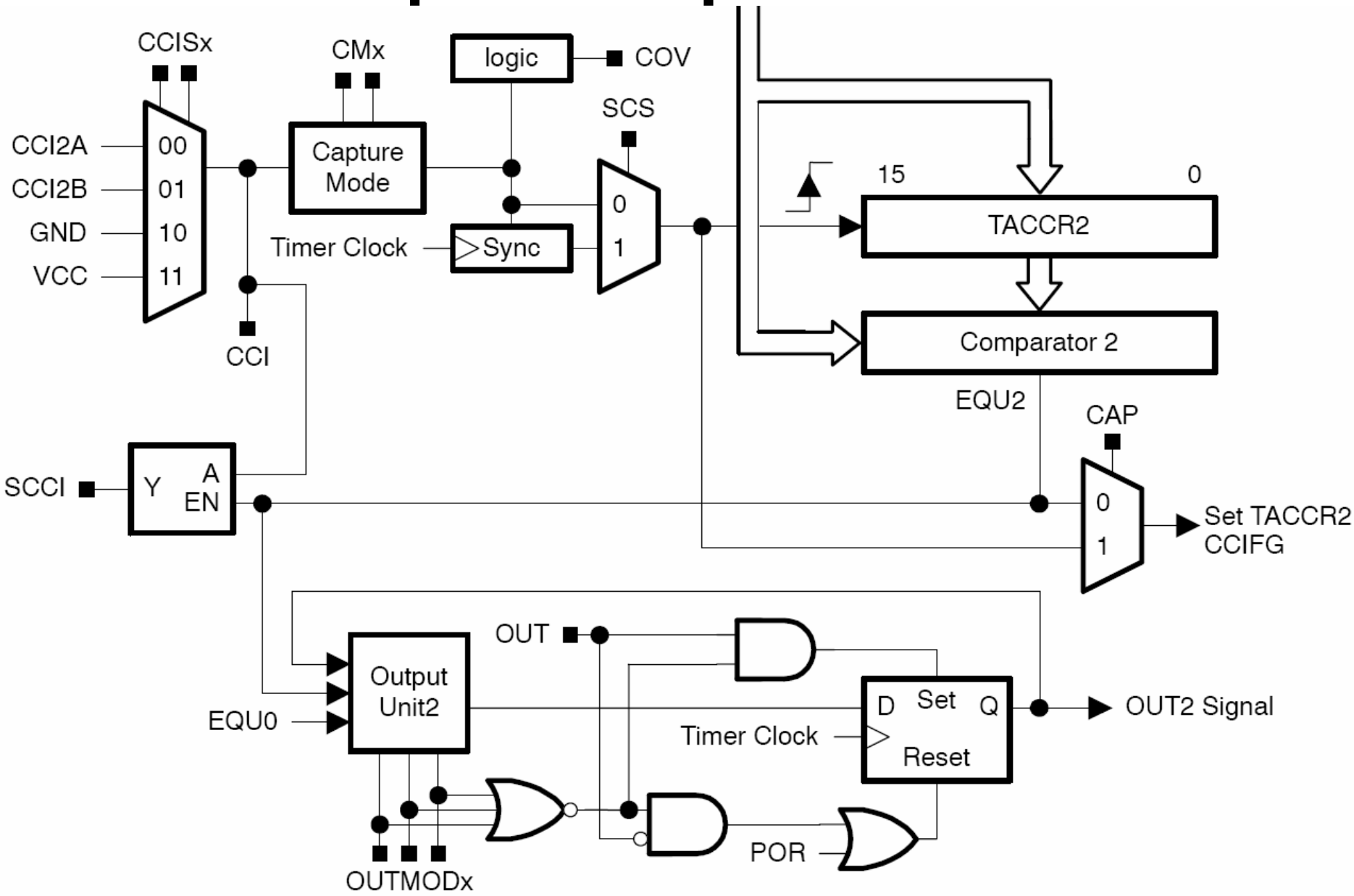
Timer Counter Block

- Counting Mode:
  - Up mode
  - Continuous mode
  - Up/down mode
- Remember **TACCR0** is special

MCx = 00 → Timer stopped
MCx = 01 → Up mode
MCx = 10 → Continuous mode
MCx = 11 → Down mode



Compare / Capture Block



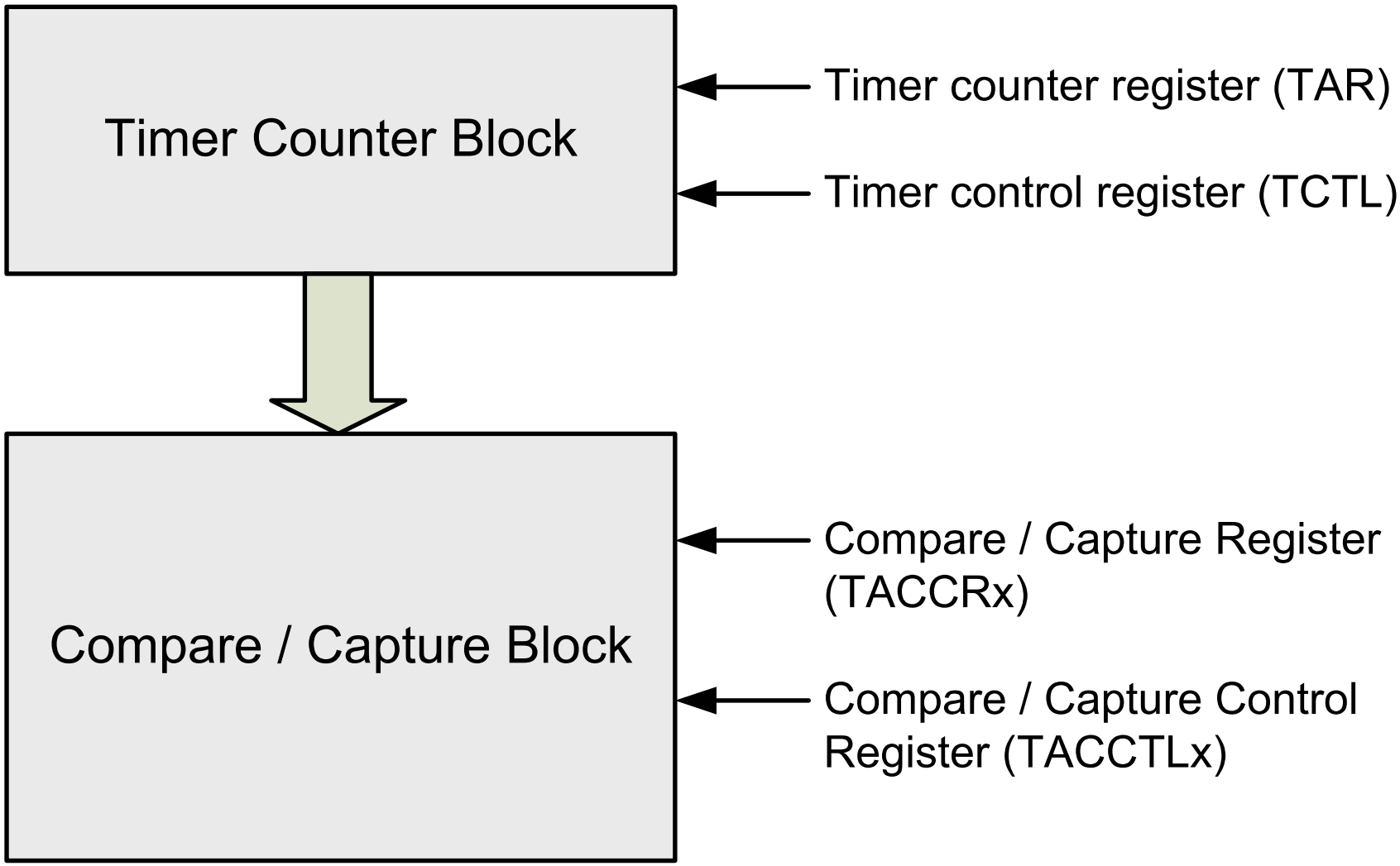
CCR Output Modes

OUTMODx	Mode	Behavior when **TAR=TACCRx**	Behavior when **TAR=TACCR0**
000	Output	Output=**OUTx**	Output=**OUTx**
001	Set	Set	Nothing
010	Toggle/Reset	Toggled	Reset
011	Set/Reset	Set	Reset
100	Toggle	Toggled	Nothing
101	Reset	Reset	Nothing
110	Toggle/Set	Toggled	Set
111	Reset/Set	Reset	Set

Timer Setup Example Code

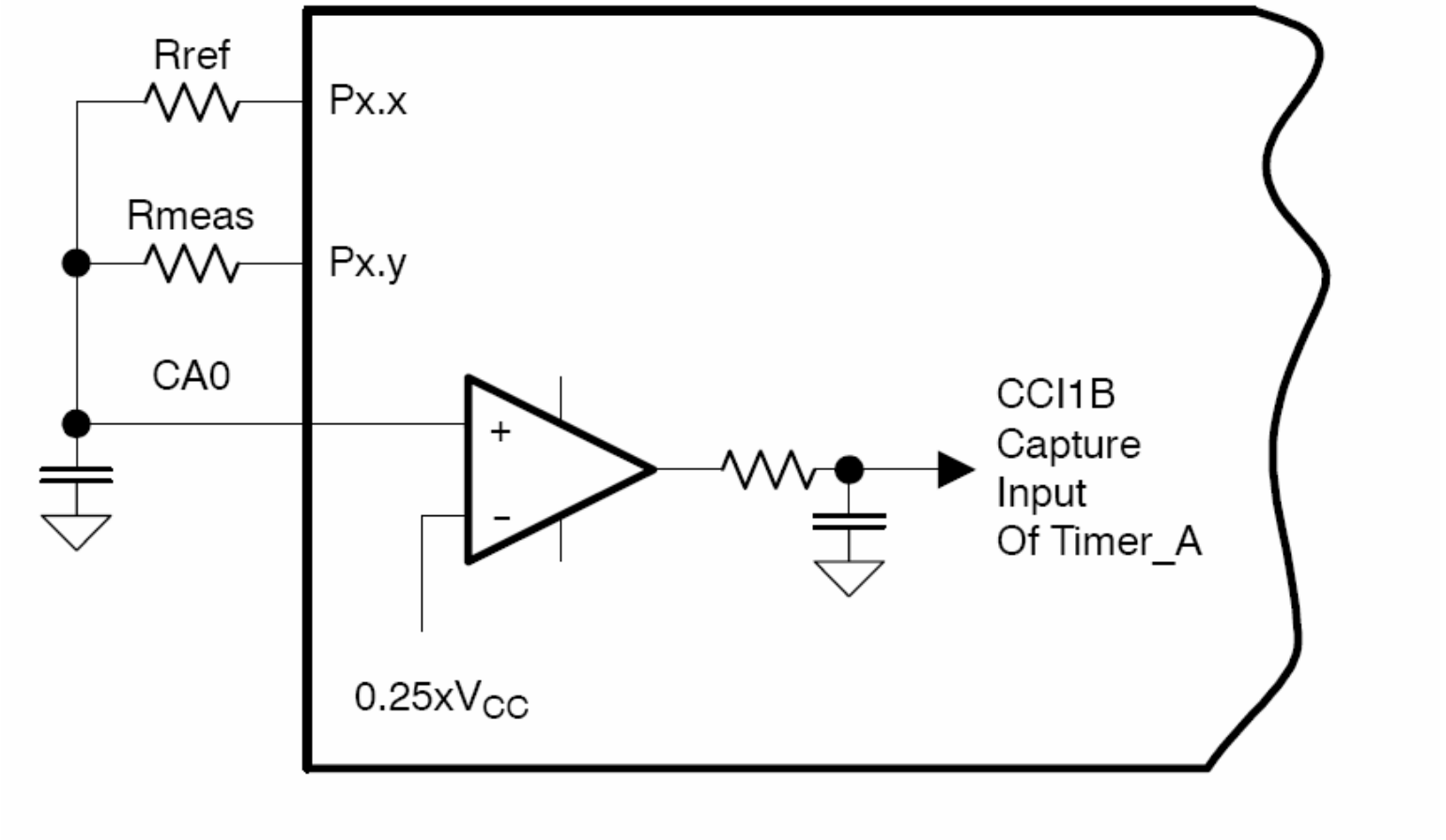
```
/*** set P1.2 and P1.3 as outputs ***/
P1DIR = BIT2 | BIT3;
/*** select P1.2 and P1.3 to be controlled by Timer A ***/
P1SEL = BIT2 | BIT3;

/*** clock source = SMCLK (4MHz), divide by 4 (1MHz) ***/
TACTL = TASSEL1 | ID1;
/*** count up to this number, then reset: ***/
TACCR0 = 5000;  // 5ms period, 200Hz
TACCR1 = 1250;  // 25% of full period
TACCR2 = 1250;  // 25% of full period

TACCTL1 = OUTMOD2 | OUTMOD1 | OUTMOD0;
TACCTL2 = OUTMOD1 | OUTMOD0;
/*** additional settings may be included here ***/

TACTL |= MC0; // start timer
```

Timer Summary

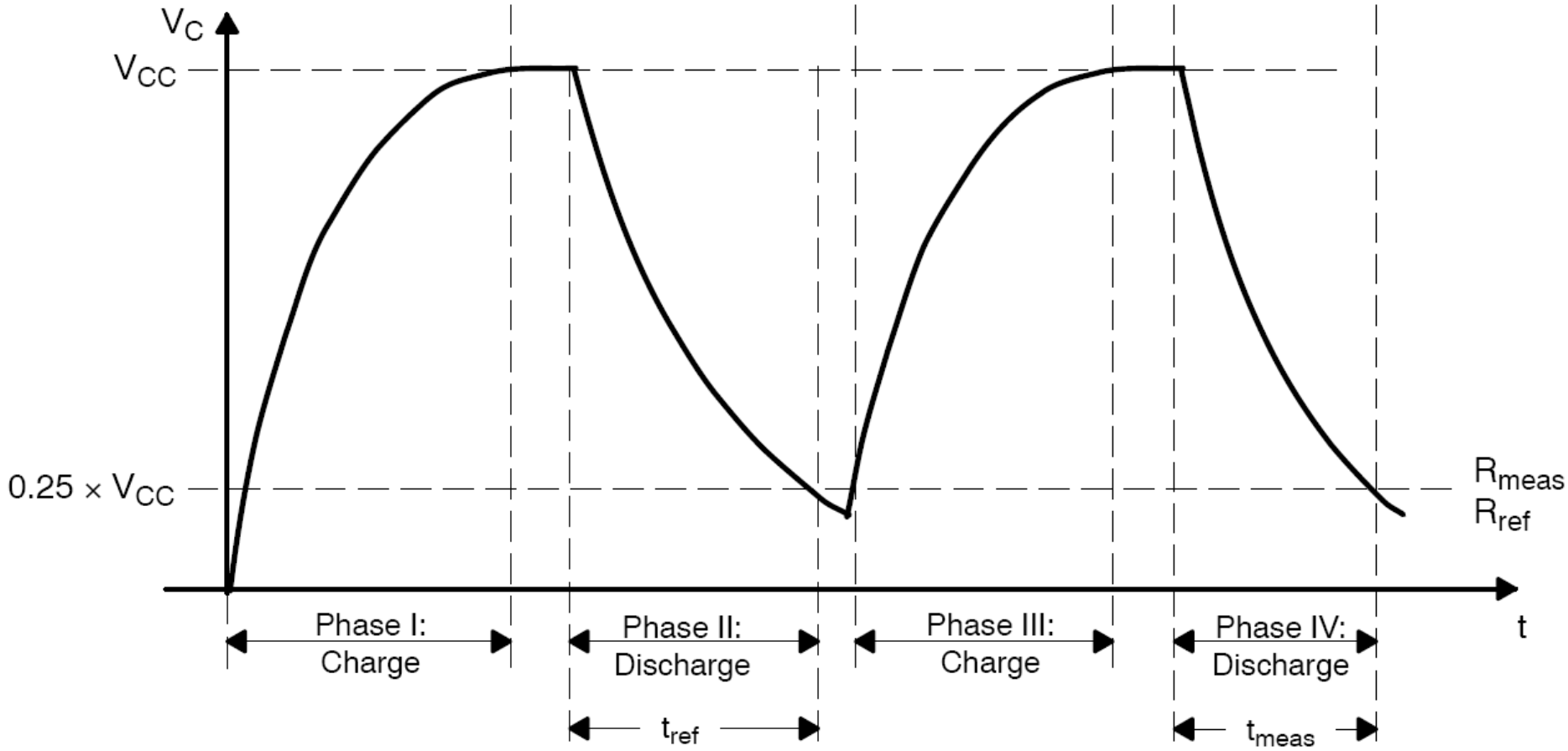


Timer Example: Thermistor

- Thermistor – temperature sensitive resistor

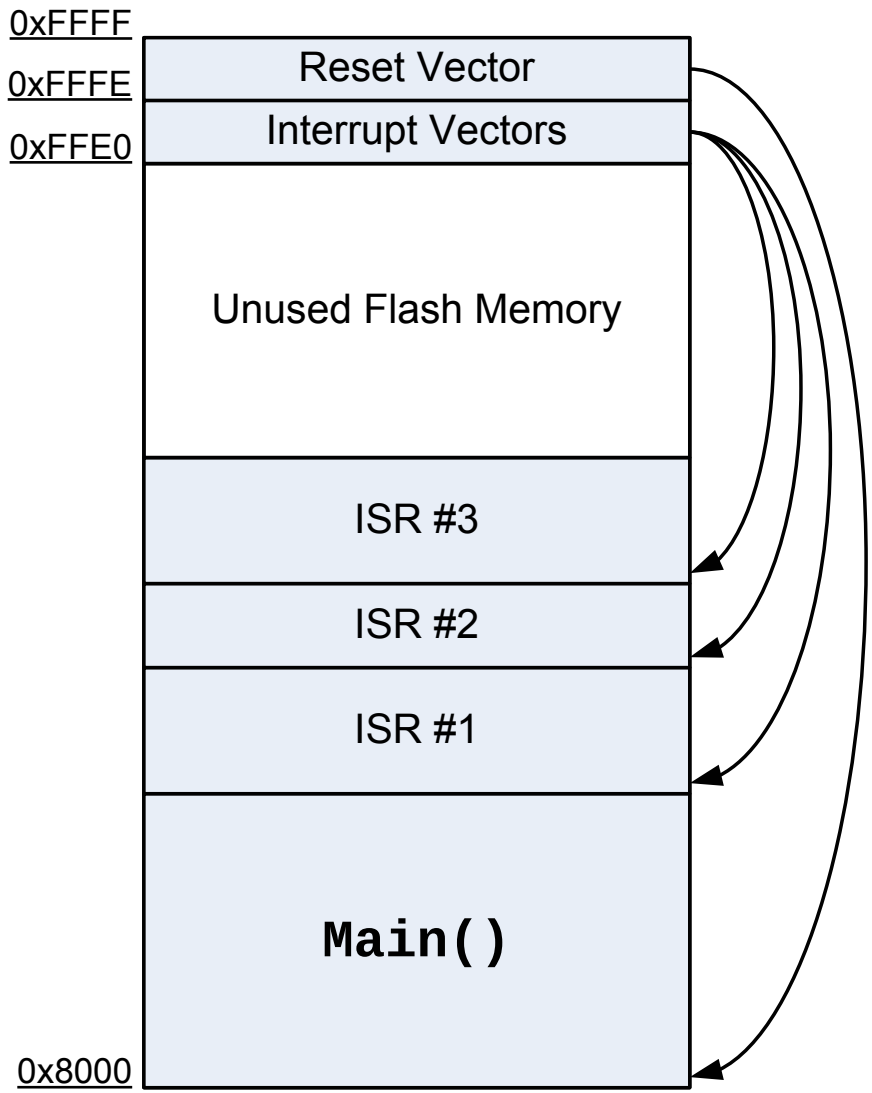


Thermistor Example Continued



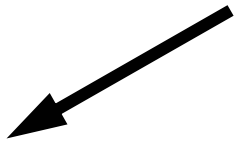
Interrupts

- Mechanism:
  - Enable interrupt
  - On interrupt event → Set interrupt flag
  - Go to Interrupt Service Routine (ISR)
  - Clear interrupt flag at the end of ISR
  - Return from interrupt
- Interrupts need to be enabled locally and globally
- **If you enable an interrupt, then you must have an ISR!**



3 Types of Interrupts

- System reset
  - Power-up
  - Watchdog
  - Flash password
- Non-maskable interrupt (NMI)
  - NMI
  - Oscillator fault
  - Flash memory access violation
- Maskable (standard)
  - ADC
  - Timer
  - I/O port
  - UART
  - etc



Beware of interrupt nesting!

Timer Interrupts

- Often, interrupts are not required
- Interrupt on overflow and when **TACCRx** = **TAR**
- Separate vectors for
  - **TACCR0**
  - Everything else
- See register maps 12-19 to 12-23 for more details

Example: Digital Port Interrupt

```
void main()
{
SAMPLE_PORT=BIT1;
P1IFG = 0x00;
P1IE = SAMPLE_PORT;
P1IES = SAMPLE_PORT; //Interrupt triggers on falling edge
...
_EINT(); //Global interrupt enable
}

#pragma vector=PORT1_VECTOR
__interrupt void PORT1_Handler (void)
{
  if ((P1IFG & SAMPLE_PORT) == SAMPLE_PORT)
    //do something here;
  P1IFG = 0x00;
}
```

Embedded Programming Styles

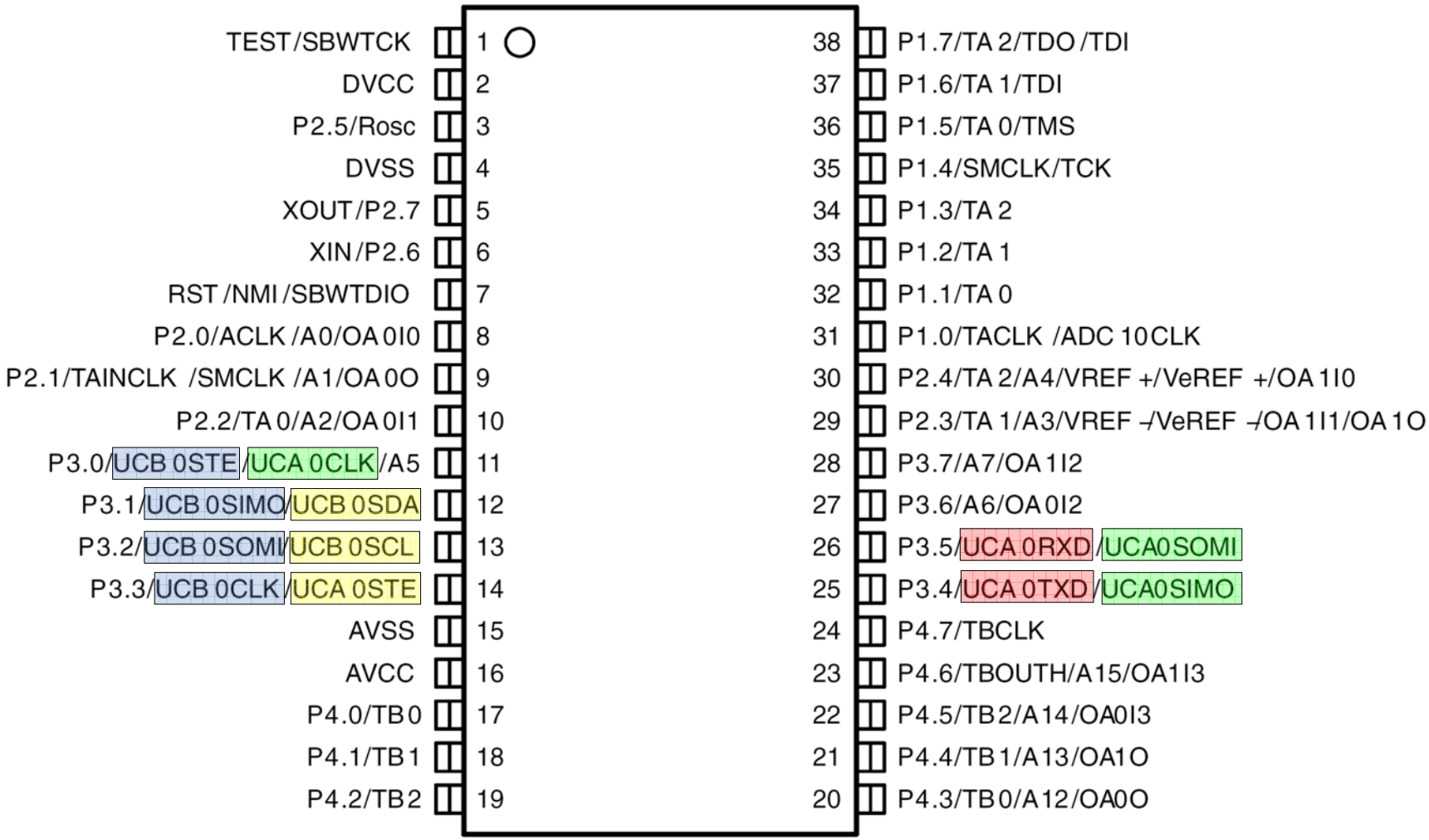
- Interrupt-driven
  - Code reside in the ISR
  - Used for handling a single interrupt source
- Event-driven
  - ISR sets flags for events
  - `main()`poll for flags and services them
  - Used for handling multiple interrupts sources

Communications

- Universal Serial Comm. Interface (USCI)
  - UCA and UCB
- UART
  - Flexible timing
  - Easy to use
  - Low speed (up to 3 Mbit/s)
- SPI
  - High speed (up to 100 Mbits/s)
  - Synchronous
- I2C
  - Low speed (up to 400 kbits/s, now 3.4 Mbits/s)
  - Multiple devices on the same bus (up to 112 nodes)

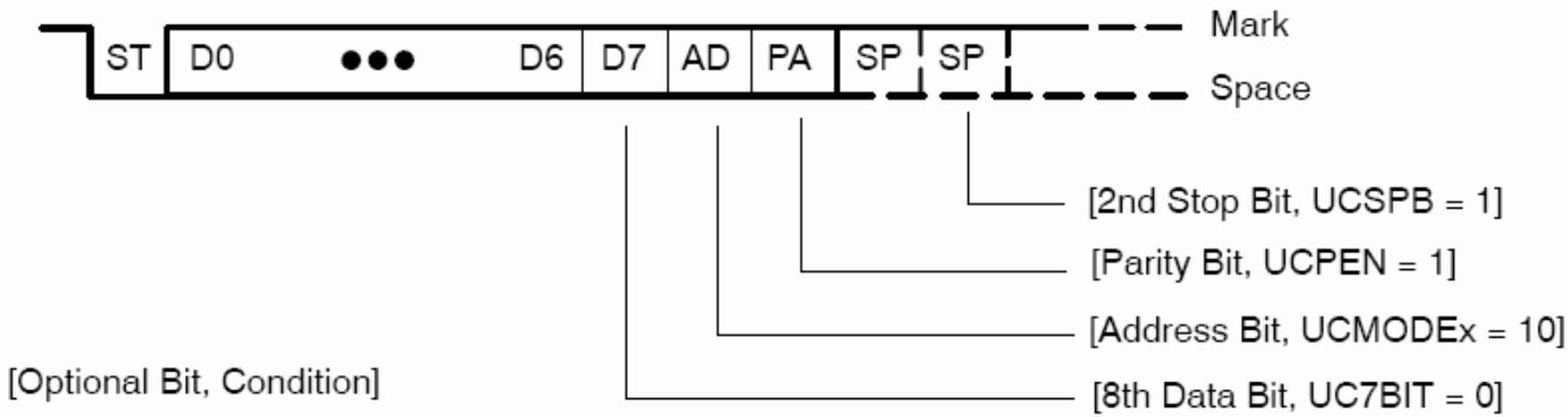
Communications Ports

MSP430x22x4 device pinout, DA package



UART Communications

- Asynchronous
- Edge triggered logic
- Some flexibility in baud rate (~ ±5%)
- Multi-device communications also possible
- Automatic baud rate detection also possible



How to Determine UART Baud Rate

- Standard baud (bits-per-second) rate
  - Multiples of 300
  - Examples: 9600, 57600, 115200, 921600
- Clock sources: ACLK, SMCLK
- **UCA0BR0** and **UCA0BR1**
  - Two 8-bit clock dividers
  - Reduce the clock source to match the standard rate
- Use modulation to reduce errors at higher speeds

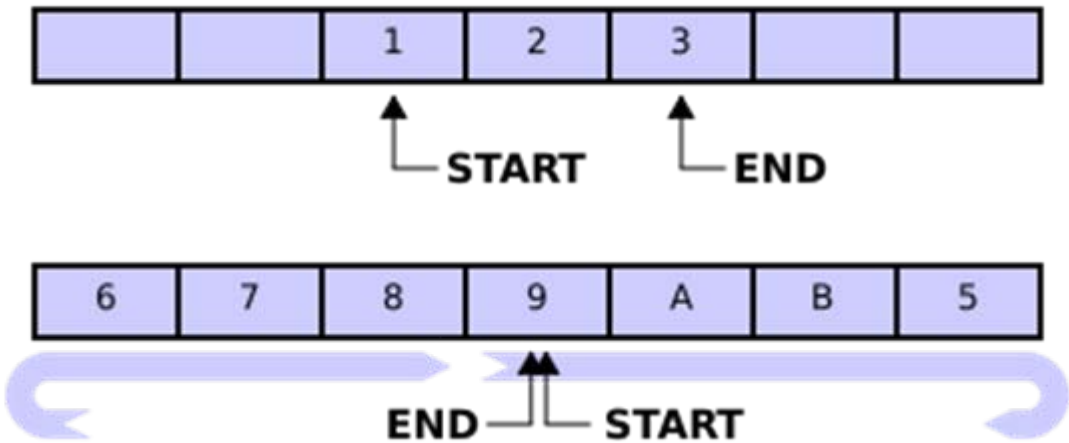
UART Registers

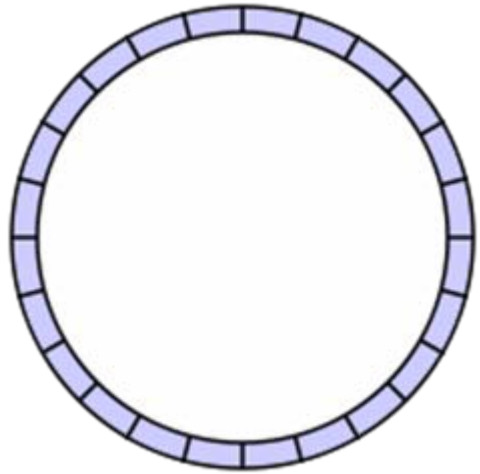
- **UCA0CTL0** and **UCA0CTL1**
  - Clock select
  - Transmit byte configuration
- **UCA0BR0** and **UCA0BR1**
  - Baud rate control
- **UCA0STAT**
  - **UCBUSY** – Tx or Rx in progress
- **UCA0RXBUF**
- **UCA0TXBUF**
- **IE2** – Interrupt enable
- **IFG2** – Interrupt flag (reset when RXBUF is read)

UART Setup Example

```
void main() {
P3SEL = BIT4 | BIT5; //Select P3.4 and P3.5 as UART controlled
P3DIR = BIT4;  // Set P3.4 as output
UCA0CTL1 = UCSSEL0; // use ACLK (16MHz) as baud rate clock
// Set up for 9600 baud
UCA0BR0 = 1666 & 0xFF; UCA0BR1 = 1666 >> 8;
IE2 |= UCA0RXIE; // enable receive data interrupt
while(1); // loop forever
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void UART_RX(void)
{
  unsigned char rx_byte;
  rx_byte = UCA0RXBUF;

  while((IFG2 & UCA0TXIFG) == 0);   // wait for TX buffer to clear
  UCA0TXBUF = rx_byte;              // echo received byte
}
```

Circular Buffer (Que)





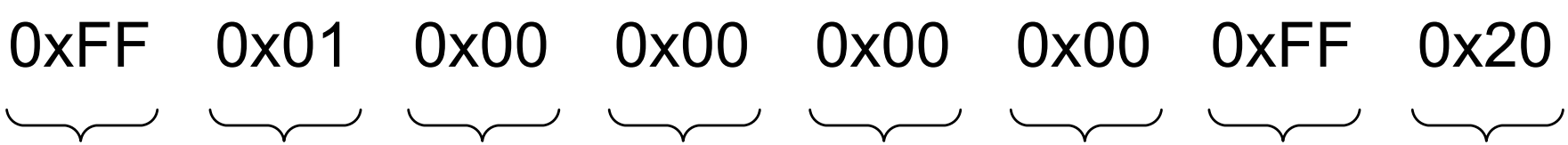
- Issues:
  - Must manage index wrap around
  - Detect buffer overflow condition

Error Checking

- CheckSum
- CRC (Cyclic Redundancy Check)
  - Look-up table version → least processor intensive

```
unsigned char tx_crc = CRC_SEED;
...

// calculate CRC
for(i = 1; i <= length; i++)
tx_crc = CRC8LUT[tx_data[i] ^ tx_crc];
tx_data[length+1] = tx_crc;
```

Standard UART Packet Format (for Lab)



- Fixed length packet
- Commands
  - 0x01: Toggle LED
  - 0x02: Set PWM duty cycle
  - 0x03: Get ADC value
- Same format for Tx and Rx packets
- Escape flags remove 0xFF bytes conflicting with the start byte