Bit manipulation

- Bit masking
- Bit flags
- Bit fields
Bit manipulation

Bit packing

- Data is commonly packed into larger unsigned integers on embedded systems
- Generally a tie in to hardware or when space is critical
  - Hardware
  - Storage
  - Binary formats
Bit manipulation

Bit packing

C1CTRL1 – dsPIC33EP256MC502
Bit manipulation

Bit masks

Example

// Abort the current CAN message transmission
C1CTRL1 = C1CTRL1 | 0x1000;

C1CTRL1 = 0x1000;
// Disable CAN message timestamping
C1CTRL1 = C1CTRL1 & 0xFFFF7;
Bit manipulation

Bit masks

Example

// Disable CAN message timestamping
CTRL1 &= ~(1 << 3);

BIT_0

BIT_1
Bit manipulation

Bit masks

• A constant that indicates which bits are relevant for a given variable
• One bits indicate significant bits
• Zero bits indicate ignore bits
Bit manipulation

Bit masks

```
#define CxCTRL1_MASK_CANCAP (1 << 3)

// Disable CAN message timestamping
C1CTRL1 &= ~CxCTRL1_MASK_CANCAP;
```

Example
Bit manipulation

Bit masking

- Setting a bit
  - ORing with 1
    
    \[ \text{c1CTRL1} \mid= \text{cXCTRL1\_MASK\_CANCAP}; \]

- Clearing a bit
  - ANDing with 0
    
    \[ \text{c1CTRL1} \&= \lnot\text{cXCTRL1\_MASK\_CANCAP}; \]

- Toggling a bit
  - XORing with 1
    
    \[ \text{c1CTRL1} \uparrow= \text{cXCTRL1\_MASK\_CANCAP}; \]
Bit manipulation

Bit masking

• Setting a bit can OR multiple masks together

Example

```c
enum {
    BUTTON_EVENT_1UP = 0x01,
    BUTTON_EVENT_2UP = 0x04
};

void example() {
    uint8_t event = BUTTON_EVENT_1UP | BUTTON_EVENT_2UP;
}
```
Bit manipulation

Bit masking

• Getting a bit
  – ANDing with 1

Example

```c
#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if (C1CTRL1 & CxCTRL1_MASK_CANCAP) == CxCTRL1_MASK_CANCAP) {
    ...
}
```
Bit manipulation

Bit masking

• Getting a bit
  – ANDing with 1

Example

#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if (ClCTRL1 & CxCTRL1_MASK_CANCAP) {
   ...
}

Bit manipulation

Example

// Retrieve the operating mode of the CAN hardware
int opmode = (ClCTRL1 & 0xE0) >> 5;
**Bit Fields** are *(unsigned)* int members of structures that occupy a specified number of adjacent bits from one to `sizeof(int)`. They may be used as an ordinary int variable in arithmetic and logical operations.

- Bit Fields:
  - Are ordinary members of a structure
  - Have a specified bit width
  - Provide bit access to a variable without masking operations
Bit Fields

- Bit Fields:
  - May only be integers (short, long, __, long long)
    - No larger than the base type
  - Unsigned by default, but may be signed
  - Non-portable across architectures/compilers!
    - Just like regular structs
Bit Fields
How to Create a Bit Field

Syntax

```c
struct StructName {
    ((un)signed) int memberName1: bitWidth;
    ...
    ((un)signed) int memberNameN: bitWidth;
}
```

Example

```c
struct ByteBits {
    unsigned int a: 1;
    long b: 1;
    short c: 2;
    unsigned d: 1;
    long long e: 3;
};
```
Bit Fields
How to Use a Bit Field

Example

typedef struct {
    unsigned int a: 1;
    long b: 1;
    short c: 2;
    unsigned d: 1;
    long long e: 3;
} ByteBits;

ByteBits x;

**bitfield struct may be declared normally or as a typedef**
Bit Fields

How to Use a Bit Field

Example

```c
struct ByteBits {
    unsigned a: 1;
    unsigned b: 1;
    unsigned c: 2;
    unsigned d: 1;
    unsigned e: 3;
} x;

int main(void)
{
    x.a = 1; // x.a may contain values from 0 to 1
    x.b = 0; // x.b may contain values from 0 to 1
    x.c = 0b10; // x.c may contain values from 0 to 3
    x.d = 0x0; // x.d may contain values from 0 to 1
    x.e = 7; // x.e may contain values from 0 to 7
}
```

Bit Fields

Microchip's SFRs

Example

// SFR register declaration
extern volatile unsigned int C1CTRL1 __attribute__((__sfr__));

// SFR bitfield declaration
typedef struct {
  unsigned WIN : 1;
  unsigned : 2;
  unsigned CANCAP : 1;
  unsigned : 1;
  unsigned OPMODE : 3;
  unsigned REQOP : 3;
  unsigned CANCKS : 1;
  unsigned ABAT : 1;
  unsigned CSIDL : 1;
} C1CTRL1BITS;
extern volatile C1CTRL1BITS C1CTRL1bits __attribute__((__sfr__));
int main(void)
{
    // Abort the current CAN message transmission
    C1CTRL1  |= 0x1000;
    C1CTRL1bits.ABAT = 1;

    // Disable CAN message timestamping
    C1CTRL1  &= 0xFFFF;
    C1CTRL1bits.CANCAP = 0;

    // If CAN message timestamping is enabled
    if (C1CTRL1 & 0x0008) {
        if (C1CTRL1bits.CANCAP) {
            ...
        }
    }
}
Bit Fields

Signed values

Example

typedef struct {
    signed int    a: 3;
    short         b: 2;
    signed short  c: 2;
    long long     d: 3;
} ByteBits;

ByteBits x;
Bit Fields
Signed values

Example

typedef struct {
    signed int    a: 3;
    short         b: 2;
    signed short  c: 1;
    long long     d: 3;
} ByteBits;

ByteBits x;
Bit Fields
Maximum bitness

Example

```c
typedef struct {
    signed int    a: 3;
    short         b: 2;
    signed short  c: 1;
    long long     d: 3;
} ByteBits;

ByteBits x;
```
Bit Fields

Maximum bitness

Example

typedef struct {
    signed short a: 3;
    short b: 2;
    signed short c: 1;
    short d: 3;
} ByteBits;

ByteBits x;
CMPE-013/L

Linked Lists

Maxwell James Dunne
typedef struct ListItem {
   struct ListItem *previousItem;
   struct ListItem *nextItem;
   char *data;
} ListItem;
Linked List

ListItem *LinkedListNew(char *data);
Linked List

List Item *LinkedListCreateAfter(List Item *item, char *data);

\[ a \rightarrow p = i; \]
\[ i \rightarrow n = a; \]
Linked List

ListItem *LinkedListCreateAfter(ListItem *item, char *data);

\( a \rightarrow n = i \rightarrow N; \)
\( a \rightarrow p = i; \)
\( o \rightarrow N \rightarrow p = a; \)
\( i \rightarrow n = a; \)
Linked List

ListItem *LinkedListCreateAfter(ListItem *item, char *data);
Linked List

char *LinkedListRemove(ListItem *item);

\[ i \rightarrow N \rightarrow p = \text{NULL}; \]
Linked List

char *LinkedListRemove(ListItem *item);

\[
\begin{array}{c}
N \\
\hline
P
\end{array}
\]

\[
\begin{array}{c}
P \\
\hline
N
\end{array}
\]

\[
\begin{array}{c}
P \\
\hline
0
\end{array}
\]

\[
\begin{array}{c}
P \\
\hline
N
\end{array}
\]

\[
i \rightarrow P \rightarrow n = 0;
\]
char *LinkedListRemove(ListItem *item);

\[ i \rightarrow p \rightarrow N = i \rightarrow N; \]
\[ i \rightarrow N \rightarrow p = i \rightarrow p; \]
Linked List

```c
ListItem *LinkedListGetFirst(ListItem *list);
```

```c
i = i -> p;
while (i -> p != NULL)
```

Linked List

int LinkedListSize(ListItem *list);

i = i; \rightarrow N;

!i = NULL