Chip Programming Model

- Modules, Registers and Bits
  - Xmega comprised of modules: CPU, SPI, UART, I2C, etc.
  - More than one instance of a module may exist.
  - Name reflects the function of the module.
  - Each module instance has a suffix to identify it:
    - USARTC0: USART 0 on port C
    - SPIE1: SPI 1 on port E
Chip Programming Model - Modules

- Each module has a fixed address in I/O memory.
- For SPI on port D:

```c
#define SPID (*(SPI_t *) 0x09C0) /*(iox64a1u.h line 2683)*/
```

<table>
<thead>
<tr>
<th>Base address</th>
<th>Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x09C0</td>
<td>SPID</td>
<td>Serial peripheral interface on port D</td>
<td>279</td>
</tr>
<tr>
<td>0x0A00</td>
<td>TCE0</td>
<td>Timer/counter 0 on port E</td>
<td>184</td>
</tr>
<tr>
<td>0x0A40</td>
<td>TCE1</td>
<td>Timer/counter 1 on port E</td>
<td>211</td>
</tr>
<tr>
<td>0x0A80</td>
<td>AWEXE</td>
<td>Advanced waveform extension on port E</td>
<td>209</td>
</tr>
<tr>
<td>0x0A90</td>
<td>HIREESE</td>
<td>High resolution extension on port E</td>
<td>211</td>
</tr>
<tr>
<td>0x0AA0</td>
<td>USART0</td>
<td>USART 0 on port E</td>
<td>300</td>
</tr>
<tr>
<td>0x0AB0</td>
<td>USART1</td>
<td>USART 1 on port E</td>
<td></td>
</tr>
<tr>
<td>0x0AC0</td>
<td>SPIE</td>
<td>Serial peripheral interface on port E</td>
<td>279</td>
</tr>
</tbody>
</table>
Chip Programming Model - Registers

- Module registers are located a fixed offset from the module base address.
- Register offsets are equal for all instances of that module type.
- Each module has status and control registers.

```c
/* (iox64a1u.h line 3584) */
/* SPID - Serial Peripheral Interface D */
#define SPID_CTRL _SFR_MEM8(0x09C0)
#define SPID_INTCTRL _SFR_MEM8(0x09C1)
#define SPID_STATUS _SFR_MEM8(0x09C2)
#define SPID_DATA _SFR_MEM8(0x09C3)
```

### 22.8 Register summary

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0x00</td>
<td>CTRL</td>
<td>CLK2X</td>
<td>ENABLE</td>
<td>DORD</td>
<td>MASTER</td>
<td>MODE[1:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+0x01</td>
<td>INTCTRL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+0x02</td>
<td>STATUS</td>
<td>IF</td>
<td>WRCOL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+0x03</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chip Programming Model - Registers

- Modules of the same type have identical sets of status and control registers.
- Otherwise, there would be multiple structs for multiple SPI modules.

```c
/* (iox64a1u.h line 2539) */
/* Serial Peripheral Interface */
typedef struct SPI_struct
{
    register8_t CTRL;     /* Control Register */
    register8_t INTCTRL;  /* Interrupt Control Register */
    register8_t STATUS;   /* Status Register */
    register8_t DATA;     /* Data Register */
} SPI_t;
```
Chip Programming Model - Bits

- Modules of the same type (SPI, USART, etc.), within the same part family (A,B,C) have identical sets of status and control bits.

![CTRL - Control register](image)

**Figure:** SPI Port B,C,D or F ControlRegs

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0x00</td>
<td>CTRLA</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>175</td>
</tr>
<tr>
<td>+0x01</td>
<td>CTRLB</td>
<td>CCDEN</td>
<td>CCCEN</td>
<td>CCBEN</td>
<td>CCAEN</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>WGMODE[2:0]</td>
<td>175</td>
</tr>
<tr>
<td>+0x02</td>
<td>CTRLC</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>CMPD</td>
<td>CMPC</td>
<td>CMPB</td>
<td>CMPA</td>
</tr>
<tr>
<td>+0x03</td>
<td>CTRLD</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>EVACT[2:0]</td>
<td>–</td>
<td>EVDLY</td>
<td>–</td>
</tr>
<tr>
<td>+0x04</td>
<td>CTRLE</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

**Figure:** Timer Counter 0 or 1 ControlRegs
Bits in a register can be have an individual function or be part of a bit group that have a joint function.

A bit with an individual function could be the single enable bit for a module. For example, bit six of the SPID control register enables that SPI device.

We would access this bit like this:

```
SPIC.CTRL = SPI_ENABLE_bm
```

The _bm suffix indicates a bit mask.

In iox64a1u.h, line 7014, we see:

```
#define SPI_ENABLE_bm 0x40 /* Enable Module bit mask. */
```

Thus, the SPID enable bit is set to one.
A bit could also be part of a bit group. The bit would be part of a group of bits that choose up to $2^n$ selections where $n$ is the number of bits in the group. For example, the group of bits that choose which of four modes a SPI module is in:

Table 22-2. SPI transfer modes.

<table>
<thead>
<tr>
<th>MODE[1:0]</th>
<th>Group configuration</th>
<th>Leading edge</th>
<th>Trailing edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>Rising, sample</td>
<td>Falling, setup</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>Rising, setup</td>
<td>Falling, sample</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>Falling, sample</td>
<td>Rising, setup</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>Falling, setup</td>
<td>Rising, sample</td>
</tr>
</tbody>
</table>

Figure: Four SPI Transfer Modes
We would access the SPI mode group configuration as:

```c
SPIC_CTRL = SPI_MODE_0_gc;
```

- The _gc suffix indicates group configuration.
- SPI_MODE_0_gc is defined as 0x00 in iox64a1u.h, line 2548.

```c
/* SPI Mode */
typedef enum SPI_MODE_enum
{
    SPI_MODE_0_gc = (0x00 << 2), /* SPI Mode 0 */
    SPI_MODE_1_gc = (0x01 << 2), /* SPI Mode 1 */
    SPI_MODE_2_gc = (0x02 << 2), /* SPI Mode 2 */
    SPI_MODE_3_gc = (0x03 << 2), /* SPI Mode 3 */
} SPI_MODE_t;
```