DISCLAIMER:
All software is provided “as is” and without any express or implied warranties, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose.

COMPATABILITY:
This software has been created and tested using the following development systems:

Compiler(s):
GCC 68HC11 compiler version 2.2

Processor(s):
Motorola 68HC11 E1,E9 operating at 2 MHz E-clock

Evaluation Board(s):
Axiom Manufacturing CMM11E1-EVBU
Axiom Manufacturing CME11E9-EVBU

Simulators(s):
68HC11 EVBU Simulator Version 0.6. PySim11 Version 0.5
(developed by Andrew Sterian)
Table of Contents

1 HC11 Microcontroller ........................................................................................................ 1
  1.1 Overview ................................................................................................................ 1
  1.2 Memory Maps ......................................................................................................... 1

2 Communication between PC and EVBU ........................................................................ 6
  2.1 Overview ................................................................................................................ 6
  2.2 Block Diagram ....................................................................................................... 7

3 Gcc 68HC11 Compiler .................................................................................................. 8
  3.1 Overview ................................................................................................................. 8
  3.2 Library Functions ................................................................................................. 8
  3.3 BUFFALO Library Functions ............................................................................... 8
  3.4 Sample Source Code ............................................................................................ 8

4 I/O Registers ................................................................................................................ 9
  4.1 Overview ................................................................................................................ 9
  4.2 Memory Mapped I/O Register ............................................................................. 9

5 Parallel I/O ................................................................................................................... 11
  5.1 Overview ............................................................................................................... 11
  5.2 Output Ports ......................................................................................................... 11
  5.3 Input Ports ............................................................................................................. 11
  5.4 Bi-directional Ports ............................................................................................. 12

6 I/O Register C Definitions ......................................................................................... 13
  6.1 Overview .............................................................................................................. 13
  6.2 Register Defines .................................................................................................. 13
  6.3 Register Usage ..................................................................................................... 13
  6.4 Bit Defines .......................................................................................................... 14
  6.5 Bit Usage .......................................................................................................... 14

7 Clocks and Counters .................................................................................................. 15
  7.1 Overview .............................................................................................................. 15
  7.2 Clocks .................................................................................................................. 15
  7.3 Free Running Clock and Counter ...................................................................... 16
    7.3.1 Overview ................................................................................................... 16
    7.3.2 Sample Source Code ............................................................................... 17
  7.4 Real-Time Interrupt Clock .................................................................................. 18
  7.5 Software Timing Measurements ........................................................................ 19
    7.5.1 Example – Measuring a Pulse Width .......................................................... 20
    7.5.2 Example – Generating a Pulse Width ......................................................... 22

8 Events and Flags ........................................................................................................ 24
  8.1 Overview ............................................................................................................... 24
  8.2 Polling Technique ............................................................................................... 25
    8.2.1 Overview ................................................................................................... 25
    8.2.2 Sample Source Code ............................................................................... 25
  8.3 Interrupt Technique ............................................................................................. 26
    8.3.1 Overview ................................................................................................... 26
    8.3.2 Sample Source Code ............................................................................... 26
List of Figures

Figure 1: 68HC11E9 Memory Map ................................................................. 2
Figure 2: CME11E9-EVBU Memory Map ....................................................... 3
Figure 3: 68HC11E1 Memory Map ................................................................. 4
Figure 4: CMM11E1-EVBU Memory Map ....................................................... 5
Figure 5: Block Diagram for Communication between PC and EVBU .............. 7
Figure 6: Block Diagram for Parallel I/O Latches .......................................... 11
Figure 7: Flow Chart for Polling Technique .................................................. 25
Figure 8: Flow Chart for Interrupt Technique .............................................. 26
Figure 9: Interrupt Vector Example ............................................................. 27
Figure 10: Interrupt Sequence of Operation .................................................. 29

List of Tables

Table 1: I/O Registers Summary ................................................................. 10
Table 2: Interrupt Vector Summary ............................................................. 28
1 HC11 Microcontroller

1.1 Overview

- Essential components packaged onto one chip
  - CPU (Motorola HC11)
  - Memory (internal)
  - I/O Peripherals (digital inputs/outputs, timers, etc)

- I/O Ports
  - External I/O signals grouped into 8-bit ports
  - There are five I/O ports (PORTA through PORTE)
  - Each port can be used as digital inputs/outputs
  - Each port also has additional functionality associated with it
    - PORTA – programmable timers
    - PORTD – serial interface
    - PORTE – analog-to-digital converter
    - PORTB/PORTC – expanded mode (address and data bus)

- There are different chip versions of the 68HC11
  - All based on the Motorola HC11 microprocessor
  - Transparent operation of I/O ports and registers
  - Versions vary in amount and type of memory and certain I/O features

- There are four operating modes
  - Operating mode determined by MODA/MODB pins on power-up
    - Single chip mode – All I/O ports available, external memory not available
    - Expanded chip mode – Not all I/O ports available, external memory available
    - Test modes – Used for factory testing and software installation

- The EVBU is an evaluation board with the microcontroller, serial hardware, external memory, and connectors. The EVBU provides a platform for quick development and testing of applications microcontroller applications

1.2 Memory Maps

- All hardware accessed by software via addresses
- Each hardware device is mapped to a certain range of addresses
- Available hardware devices
  - Internal memory
  - External memory
  - I/O registers

- Additional hardware devices can be connected and mapped to the microcontroller using address decoding.
Figure 1: 68HC11E9 Memory Map
<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
<th>Address Range</th>
<th>Description</th>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000-$01FF</td>
<td>Internal RAM (512 Bytes)</td>
<td>$0000-$01FF</td>
<td>External RAM (32 k Bytes)</td>
<td>$0000-$01FF</td>
<td>Internal RAM</td>
</tr>
<tr>
<td>$01FF-$0200</td>
<td>Not Used</td>
<td>$0200-$03F</td>
<td>I/O Registers (64 Bytes)</td>
<td>$0200-$03F</td>
<td>External RAM</td>
</tr>
<tr>
<td>$03F-$040</td>
<td>Not Used</td>
<td>$040-$0FF</td>
<td>Not Used</td>
<td>$040-$0FF</td>
<td>I/O Registers</td>
</tr>
<tr>
<td>$0FF-$103F</td>
<td>Not Used</td>
<td>$103F-$1040</td>
<td>Not Used</td>
<td>$103F-$1040</td>
<td>External RAM</td>
</tr>
<tr>
<td>$1040-$10FF</td>
<td>Internal EEPROM (512 Bytes)</td>
<td>$10FF-$B5FF</td>
<td>Internal EEPROM (512 Bytes)</td>
<td>$10FF-$B5FF</td>
<td>Not Used</td>
</tr>
<tr>
<td>$B5FF-$B600</td>
<td>Not Used</td>
<td>$B600-$B7FF</td>
<td>Not Used</td>
<td>$B600-$B7FF</td>
<td>Internal EEPROM (512 Bytes)</td>
</tr>
<tr>
<td>$B7FF-$B800</td>
<td>Internal EPROM (12 k Bytes)</td>
<td>$B800-$CFFF</td>
<td>Not Used</td>
<td>$B800-$CFFF</td>
<td>Not Used</td>
</tr>
<tr>
<td>$CFFF-$D000</td>
<td>Not Used</td>
<td>$D000-$DFFF</td>
<td>External ROM (8 k Bytes)</td>
<td>$D000-$DFFF</td>
<td>Internal EPROM (12 k Bytes)</td>
</tr>
<tr>
<td>$DFFF-$E000</td>
<td>External ROM (8 k Bytes)</td>
<td>$E000-$FFFF</td>
<td>Not used</td>
<td>$E000-$FFFF</td>
<td>Not used</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Not used</td>
<td>$FFFF</td>
<td>Not used</td>
<td>$FFFF</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Figure 2: CME11E9-EVBU Memory Map
### 68HC11 Memory Map

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Memory Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000 - $01FF</td>
<td>Internal RAM (512 Bytes)</td>
<td></td>
</tr>
<tr>
<td>$01FF - $0200</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>$0200 - $0FFFF</td>
<td>Stack Space (21+ Bytes)</td>
<td></td>
</tr>
<tr>
<td>$0FFFF</td>
<td>Monitor Variable Space  (121 Bytes)</td>
<td></td>
</tr>
<tr>
<td>$1000 - $103F</td>
<td>I/O Registers (64 Bytes)</td>
<td></td>
</tr>
<tr>
<td>$103F - $1040</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>$B5FF - $B600</td>
<td>Internal EEPROM (512 Bytes)</td>
<td></td>
</tr>
<tr>
<td>$B600</td>
<td>User Program Space</td>
<td>(512 Bytes)</td>
</tr>
<tr>
<td>$B7FF - $B800</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>$B800</td>
<td>User Program Space</td>
<td>(256 Bytes)</td>
</tr>
<tr>
<td>$B800 - $FFFF</td>
<td>User Data Space</td>
<td>(54- Bytes)</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Stack Space</td>
<td>(21+ Bytes)</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Monitor Variable Space</td>
<td>(121 Bytes)</td>
</tr>
</tbody>
</table>

*Figure 3: 68HC11E1 Memory Map*
### 68HC11E1 Mapping

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>Internal RAM (512 Bytes)</td>
</tr>
<tr>
<td>$01FF</td>
<td>Not Used</td>
</tr>
<tr>
<td>$0200</td>
<td></td>
</tr>
<tr>
<td>$0FFF</td>
<td>I/O Registers (64 Bytes)</td>
</tr>
<tr>
<td>$1000</td>
<td>Not Used</td>
</tr>
<tr>
<td>$103F</td>
<td></td>
</tr>
<tr>
<td>$1040</td>
<td></td>
</tr>
<tr>
<td>$B5FF</td>
<td>Internal EEPROM (512 Bytes)</td>
</tr>
<tr>
<td>$B600</td>
<td>Not Used</td>
</tr>
<tr>
<td>$B7FF</td>
<td></td>
</tr>
<tr>
<td>$B800</td>
<td></td>
</tr>
<tr>
<td>$FFFF</td>
<td></td>
</tr>
</tbody>
</table>

### CMM11E1 Mapping

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>External RAM (32 k Bytes)</td>
</tr>
<tr>
<td>$01FF</td>
<td>Not Used</td>
</tr>
<tr>
<td>$0200</td>
<td></td>
</tr>
<tr>
<td>$0FFF</td>
<td>I/O Registers (64 Bytes)</td>
</tr>
<tr>
<td>$1000</td>
<td>Not Used</td>
</tr>
<tr>
<td>$103F</td>
<td></td>
</tr>
<tr>
<td>$1040</td>
<td></td>
</tr>
<tr>
<td>$7FFF</td>
<td>External EEPROM (32 k Bytes)</td>
</tr>
<tr>
<td>$8000</td>
<td>Not Used</td>
</tr>
<tr>
<td>$B5FF</td>
<td></td>
</tr>
<tr>
<td>$B600</td>
<td></td>
</tr>
<tr>
<td>$B7FF</td>
<td></td>
</tr>
<tr>
<td>$B800</td>
<td></td>
</tr>
<tr>
<td>$DFFF</td>
<td>Monitor Program</td>
</tr>
<tr>
<td>$E000</td>
<td></td>
</tr>
<tr>
<td>$FFFF</td>
<td></td>
</tr>
</tbody>
</table>

### Overall Mapping

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>Internal RAM</td>
</tr>
<tr>
<td>$01FF</td>
<td>Not Used</td>
</tr>
<tr>
<td>$0200</td>
<td>External RAM</td>
</tr>
<tr>
<td>$0FFF</td>
<td>I/O Registers</td>
</tr>
<tr>
<td>$1000</td>
<td>Not Used</td>
</tr>
<tr>
<td>$103F</td>
<td>External RAM</td>
</tr>
<tr>
<td>$1040</td>
<td>I/O Registers</td>
</tr>
<tr>
<td>$7FFF</td>
<td>Not Used</td>
</tr>
<tr>
<td>$8000</td>
<td>External EEPROM</td>
</tr>
<tr>
<td>$B5FF</td>
<td>Internal EEPROM</td>
</tr>
<tr>
<td>$B600</td>
<td>Not Used</td>
</tr>
<tr>
<td>$B7FF</td>
<td>External EEPROM</td>
</tr>
<tr>
<td>$B800</td>
<td>I/O Registers</td>
</tr>
<tr>
<td>$DFFF</td>
<td>Monitor Program</td>
</tr>
<tr>
<td>$E000</td>
<td>Not Used</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Monitor Program</td>
</tr>
</tbody>
</table>

**Figure 4: CMM11E1-EVBU Memory Map**
2 Communication between PC and EVBU

2.1 Overview

- Monitor program
  - Stored in permanent memory
  - Provides easy access to memory/hardware on the microcontroller
  - The monitor program on the EVBU is called BUFFALO
  - A special Wallace monitor program was written for the Wallace robots.
    - The Wallace monitor program allows easy access to the I/O on the robot
    - BUFFALO provides more general access to the I/O on the 68HC11

- Standard Communication
  - On the PC, the standard input device is the keyboard and the standard output device is the monitor.
  - The serial port on the EVBU is used as the standard input and output device
  - The monitor programs use the standard I/O for communicating
  - The PC must be connected to the EVBU via a serial cable
  - A terminal emulation program (such as HyperTerminal) can be used on the PC to transmit and receive data across the serial port
  - The communication protocols on the PC and EVBU must match
    - 9600 baud, 8 data bits, no parity, 1 stop bit

- Loading a program
  - The executable code must be copied from the host computer (PC) to the memory on the target computer (microcontroller/EVBU)
  - The S19 file format is not true executable code
    - The S19 file is in a special format for transmitting to BUFFALO
    - BUFFALO translates the S19 format into executable code and stores it in the appropriate memory locations
  - To load an S19 file:
    - Use the LOAD T command to prepare BUFFALO to receive an S19 file
    - Transmit the S19 as a text file

- Executing a program
  - The 68HC11 is a serial processor (it can only execute one program at a time)
  - To execute a program
    - Use the GO 1040 command to transfer program execution from BUFFALO to the beginning of your program.
    - When your program ends, program execution will automatically be transferred back to BUFFALO
2.2 Block Diagram

![Block Diagram for Communication between PC and EVBU]

**Figure 5: Block Diagram for Communication between PC and EVBU**
3 Gcc 68HC11 Compiler

3.1 Overview

- Gcc 68HC11 compiler used to compile C code for the 68HC11 microcontroller.
- It is a 16 bit compiler (an integer/word is 16 bits)
- The 68HC11 is not a floating-point processor. The compiler can handle floating-point math but the code becomes large and time consuming. Do not use floating-point variables or math.
- Use the GCC 68HC11 DOS Prompt and gcc6811 batch file to compile programs.
- Compiler creates an S19 file.

  gcc6811 test.c

- Compiler and linker options pre-configured in gcc6811 batch file.
- Default starting address for programs is 1040.

3.2 Library Functions

- You cannot use most of the C library functions you are familiar with. The functions are either not available, require too much memory, and/or consume too much time.
- Unable to use standard I/O functions (printf, scanf, etc.).
- All library functions that you use become part of your program and must be downloaded into memory on the EVBU.

3.3 BUFFALO Library Functions

- BUFFALO program organized into subroutines (modular programming)
- BUFFALO already stored in ROM on the EVBU
- You can access BUFFALO subroutines if you know the address and parameter format
- Use Buffalo library functions to take advantage of BUFFALO subroutines
- Refer to BUFFALO Library Documentation (include buffalo.h header file)

3.4 Sample Source Code

  standard.c (DOS)
  buffalo.c (CMM11E1, CME11E9, simulator)
  int.c (CMM11E1, CME11E9, simulator)
  float.c (CMM11E1, CME11E9, simulator)
4 I/O Registers

4.1 Overview

- The I/O registers are used to control the I/O features on the 68HC11.
- Each register has a specific purpose.
- The bits in some registers also have a specific purpose.
- Some registers can be modified by the user while others cannot.
- The registers are accessed via memory mapping.

4.2 Memory Mapped I/O Register

- The I/O registers on the 68HC11 are mapped to specific memory locations.
- Read memory location to determine register value.
- Write to memory location to change register value (if it can be changed).
<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>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>PORTA</td>
<td>PA7</td>
<td>PA6</td>
<td>PA5</td>
<td>PA4</td>
<td>PA3</td>
<td>PA2</td>
<td>PA1</td>
<td>PA0</td>
</tr>
<tr>
<td>1001</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>PORTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>PORTD</td>
<td>PC7</td>
<td>PC6</td>
<td>PC5</td>
<td>PC4</td>
<td>PC3</td>
<td>PC2</td>
<td>PC1</td>
<td>PC0</td>
</tr>
<tr>
<td>1004</td>
<td>PORTB</td>
<td>PB7</td>
<td>PB6</td>
<td>PB5</td>
<td>PB4</td>
<td>PB3</td>
<td>PB2</td>
<td>PB1</td>
<td>PB0</td>
</tr>
<tr>
<td>1005</td>
<td>PORTCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>DDRC</td>
<td>DDC7</td>
<td>DDC6</td>
<td>DDC5</td>
<td>DDC4</td>
<td>DDC3</td>
<td>DDC2</td>
<td>DDC1</td>
<td>DDC0</td>
</tr>
<tr>
<td>1008</td>
<td>PORTD</td>
<td>0</td>
<td>0</td>
<td>PD5</td>
<td>PD4</td>
<td>PD3</td>
<td>PD2</td>
<td>PD1</td>
<td>PD0</td>
</tr>
<tr>
<td>1009</td>
<td>DDRD</td>
<td>0</td>
<td>0</td>
<td>DDD5</td>
<td>DDD4</td>
<td>DDD3</td>
<td>DDD2</td>
<td>DDD1</td>
<td>DDD0</td>
</tr>
<tr>
<td>100A</td>
<td>PORTE</td>
<td>PE7</td>
<td>PE6</td>
<td>PE5</td>
<td>PE4</td>
<td>PE3</td>
<td>PE2</td>
<td>PE1</td>
<td>PE0</td>
</tr>
<tr>
<td>100B</td>
<td>CFORC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100C</td>
<td>OC1M</td>
<td>OC1M7</td>
<td>OC1M6</td>
<td>OC1M5</td>
<td>OC1M4</td>
<td>OC1M3</td>
<td>OC1M2</td>
<td>OC1M1</td>
<td>OC1M0</td>
</tr>
<tr>
<td>100D</td>
<td>OC1D</td>
<td>OC1D7</td>
<td>OC1D6</td>
<td>OC1D5</td>
<td>OC1D4</td>
<td>OC1D3</td>
<td>OC1D2</td>
<td>OC1D1</td>
<td>OC1D0</td>
</tr>
<tr>
<td>100E</td>
<td>TCNT(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>100F</td>
<td>TCNT(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>TIC1(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>1011</td>
<td>TIC1(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1012</td>
<td>TIC2(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>1013</td>
<td>TIC2(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1014</td>
<td>TIC3(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>1015</td>
<td>TIC3(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1016</td>
<td>TOC1(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>1017</td>
<td>TOC1(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1018</td>
<td>TOC2(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>1019</td>
<td>TOC2(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101A</td>
<td>TOC3(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>101B</td>
<td>TOC3(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101C</td>
<td>TOC4(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>101D</td>
<td>TOC4(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101E</td>
<td>T14O5(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>101F</td>
<td>T14O5(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1020</td>
<td>TCTL1</td>
<td>OM2</td>
<td>OL2</td>
<td>OM3</td>
<td>OL3</td>
<td>OM4</td>
<td>OL4</td>
<td>OM5</td>
<td>OL5</td>
</tr>
<tr>
<td>1021</td>
<td>TCTL2</td>
<td>EDG4B</td>
<td>EDG4A</td>
<td>EDG3B</td>
<td>EDG3A</td>
<td>EDG2B</td>
<td>EDG2A</td>
<td>EDG1B</td>
<td>EDG1A</td>
</tr>
<tr>
<td>1022</td>
<td>TMsk1</td>
<td>OC1I</td>
<td>OC2I</td>
<td>OC3I</td>
<td>OC4I</td>
<td>I405I</td>
<td>I405F</td>
<td>I405F</td>
<td>I405F</td>
</tr>
<tr>
<td>1023</td>
<td>TFLG1</td>
<td>OC1F</td>
<td>OC2F</td>
<td>OC3F</td>
<td>OC4F</td>
<td>I405F</td>
<td>I405F</td>
<td>I405F</td>
<td>I405F</td>
</tr>
<tr>
<td>1024</td>
<td>TMsk2</td>
<td>TO1</td>
<td>RT11</td>
<td>PAOVI</td>
<td>PAI1</td>
<td>0</td>
<td>0</td>
<td>PR1</td>
<td>PR0</td>
</tr>
<tr>
<td>1025</td>
<td>TFLG2</td>
<td>TOF</td>
<td>RT1F</td>
<td>PAOVF</td>
<td>PAIF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1026</td>
<td>PACTL</td>
<td>DDRA7</td>
<td>PAEN</td>
<td>PAMOD</td>
<td>PEDGE</td>
<td>DDRA3</td>
<td>I405S</td>
<td>RTR1</td>
<td>RTR0</td>
</tr>
<tr>
<td>1027</td>
<td>PACNT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1028</td>
<td>SPCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1029</td>
<td>SPSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102A</td>
<td>SPCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102B</td>
<td>BAUD</td>
<td>TCLR</td>
<td>0</td>
<td>SCP1</td>
<td>SCP0</td>
<td>RCKB</td>
<td>SCR2</td>
<td>SCR1</td>
<td>SCR0</td>
</tr>
<tr>
<td>102C</td>
<td>SCCR1</td>
<td>R8</td>
<td>T8</td>
<td>0</td>
<td>M</td>
<td>WAKE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>102D</td>
<td>SCCR2</td>
<td>TIE</td>
<td>TCIE</td>
<td>RIE</td>
<td>ILIE</td>
<td>REN</td>
<td>RE</td>
<td>REN</td>
<td>REN</td>
</tr>
<tr>
<td>102E</td>
<td>SC5R</td>
<td>TREG</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>IL</td>
<td>NF</td>
<td>FE</td>
<td>FE</td>
</tr>
<tr>
<td>102F</td>
<td>SCDR</td>
<td>R7/T7</td>
<td>R6/T6</td>
<td>R5/T5</td>
<td>R4/T4</td>
<td>R3/T3</td>
<td>R2/T2</td>
<td>R1/T1</td>
<td>R0/T0</td>
</tr>
<tr>
<td>1030</td>
<td>AIXT1</td>
<td>CCF</td>
<td>0</td>
<td>SCAN</td>
<td>MULT</td>
<td>CD</td>
<td>C2</td>
<td>CB</td>
<td>CA</td>
</tr>
<tr>
<td>1031</td>
<td>ADR1</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1032</td>
<td>ADR2</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1033</td>
<td>ADR3</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1034</td>
<td>ADR4</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1035</td>
<td>BPROT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1036</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1037</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1038</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1039</td>
<td>OPTION</td>
<td>ADPU</td>
<td>CSEL</td>
<td>IRQE</td>
<td>DLY</td>
<td>CME</td>
<td>0</td>
<td>CR1</td>
<td>CR0</td>
</tr>
<tr>
<td>103A</td>
<td>COPRST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103B</td>
<td>PPROM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103C</td>
<td>MPRI0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103D</td>
<td>INIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103E</td>
<td>TEST1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103F</td>
<td>CONFIG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5 Parallel I/O

5.1 Overview
- Access digital inputs and outputs via corresponding PORT registers.
- Each signal/pin has corresponding bit in PORT register.
- Determine current input and output signal values by reading PORT registers.
- Assign output signal values by writing to PORT registers.
- You cannot change the value of an input signal.

5.2 Output Ports
- Write to I/O register to assign output values.
- Read from I/O register to determine output values.

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</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>
</tr>
</thead>
<tbody>
<tr>
<td>1004</td>
<td>PORTB</td>
<td>PB7</td>
<td>PB6</td>
<td>PB5</td>
<td>PB4</td>
<td>PB3</td>
<td>PB2</td>
<td>PB1</td>
<td>PB0</td>
</tr>
</tbody>
</table>

5.3 Input Ports
- Read from I/O register to determine input values.
- Writing to I/O register has no effect.

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</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>
</tr>
</thead>
<tbody>
<tr>
<td>100A</td>
<td>PORTE</td>
<td>PE7</td>
<td>PE6</td>
<td>PE5</td>
<td>PE4</td>
<td>PE3</td>
<td>PE2</td>
<td>PE1</td>
<td>PE0</td>
</tr>
</tbody>
</table>
5.4 Bi-directional Ports

- Configure direction using data direction (DD) bits. Reset (default) value is 0 (input).
  
  0 = input
  
  1 = output
- Data direction bits grouped into data direction registers (DDR). The DD bits correspond with the bits/pins they control.
- Read from I/O register to determine pin values.
- Write to I/O register to set output pins (input pins are not effected).

### Port C Summary (single chip mode)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</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>
</tr>
</thead>
<tbody>
<tr>
<td>1007</td>
<td>DDRC</td>
<td>DDC7</td>
<td>DDC6</td>
<td>DDC5</td>
<td>DDC4</td>
<td>DDC3</td>
<td>DDC2</td>
<td>DDC1</td>
<td>DDC0</td>
</tr>
<tr>
<td>1003</td>
<td>PORTC</td>
<td>PC7</td>
<td>PC6</td>
<td>PC5</td>
<td>PC4</td>
<td>PC3</td>
<td>PC2</td>
<td>PC1</td>
<td>PC0</td>
</tr>
</tbody>
</table>

### Port D Summary

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</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>
</tr>
</thead>
<tbody>
<tr>
<td>1009</td>
<td>DDRD</td>
<td>0</td>
<td>0</td>
<td>DDD5</td>
<td>DDD4</td>
<td>DDD3</td>
<td>DDD2</td>
<td>DDD1</td>
<td>DDD0</td>
</tr>
<tr>
<td>1008</td>
<td>PORTD</td>
<td>0</td>
<td>0</td>
<td>PD5</td>
<td>PD4</td>
<td>PD3</td>
<td>PD2</td>
<td>PD1</td>
<td>PD0</td>
</tr>
</tbody>
</table>

### Port A Summary

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</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>
</tr>
</thead>
<tbody>
<tr>
<td>1026</td>
<td>PACTL</td>
<td>DDRA7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>DDRA3</td>
<td>I4/O5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1000</td>
<td>PORTA</td>
<td>PA7</td>
<td>PA6</td>
<td>PA5</td>
<td>PA4</td>
<td>PA3*</td>
<td>PA2</td>
<td>PA1</td>
<td>PA0</td>
</tr>
</tbody>
</table>

*PA3 has two control pins. If the I4/O5 bit is equal to 0 (default) the PA3 pin is configured as an output. The OM5 and OL5 bits control the output state. If OC5 is disabled (OM5 = 0 and OL5 = 0) or if PA3 is configured for input capture (I4/O5 = 1) then the DDA3 bit determines the direction of the PA3 pin.
6 I/O Register C Definitions

6.1 Overview

- The hc11e9.h header file contains the definitions for the 68HC11 I/O registers.

```c
#include <hc11e9.h>
```

6.2 Register Defines

- I/O registers are defined as a de-referenced pointer to their corresponding address.
- You do not need to place this in your source code (defined in hc11e9.h header file)

```c
#define PORTA *(volatile unsigned char *)0x1000
#define PORTB *(unsigned char *)0x1004
```

6.3 Register Usage

- Use I/O registers in your program like variables.
- You can use them in assignments, conditional statements, etc.
- You can always change the value of a variable. They do not change unless you change them.
- You cannot always change the value of an I/O register. Some of them can change on their own.

```c
int x, y;
```

```c
x = PORTB;
y = PORTB;
PORTB = 0xE5;
if (PORTB == 0x40)
...
```

- Variables x and y will always be the same
  - Bits (pins) PB7, PB6, PB5, PB2, PB0 set to a 1 (high)
  - Bits (pins) PB4, PB3, PB1 set to a zero (low)
  - Checks if PB6 = 1 and all other pins = 0

```c
x = PORTE;
y = PORTE;
PORTE = 0xE5;
if (PORTE == 0x40)
...
```

- Variables x and y may be different
  - Has no effect on bits (pins)
  - Checks if PE6 = 1 and all other pins = 0
6.4 Bit Defines

- I/O register bits are defined as the corresponding bit mask.
- I/O register bit defines are not associated with specific registers (all bit 7 bits defined the same). You must ensure correct bit/register combination usage.
- Use appropriate bit defines instead of actual mask to make code easier to understand.
- You do not need to place this in your source code (defined in `hc11e9.h` header file)

```c
#define bit7 0x80
#define bit6 0x40
#define PB7 bit7
#define PB6 bit6
#define PE7 bit7
#define PE6 bit6
```

6.5 Bit Usage

- You cannot access bits directly. You must go through the I/O register.
- The bit defines are not variables. They do not represent the value of the bit.
- The actual bit values are in the I/O register. Use the bit defines and bit wise operations to access specific bits in the I/O register.

```c
/* Invalid C statements */
PB7 = 1;    /* Same as 0x80 = 1 */
P7 = 0;     /* Same as 0x80 = 0 */

/* Modify entire register */
PORTB = 0x04;    /* Cryptic */
PORTB = PB2;     /* More understandable */
PORTB = (PB3 | PB4);  /* Combine masks */

/* Set specific bits to 1, leave other bits alone */
PORTB = PORTB | PB3;    /* PB3 = 1 */
PORTB |= PB3;         /* Shortcut notation */
PORTB |= (PB3 | PB4);  /* PB3 = 1 , PB4 = 1 */

/* Set specific bits to 0, leave other bits alone */
PORTB = PORTB & ~PB3;  /* PB3 = 0 */
PORTB &= ~PB3;        /* Shortcut notation */
PORTB &= ~(PB3 | PB4); /* PB3 = 0 , PB4 = 0 */

/* Check specific bits */
if ( (PORTB & PB3) == 0 ) ... /* PB3 = 0 ? */
if ( (PORTB & PB3) == PB3 ) ... /* PB3 = 1 ? */
```

**IMPORTANT NOTE:**

- When referring to bits (pins) outside the context of C code, it is valid to say PB7 = 1.
- When referring to bits in the context of C code, it is **invalid** to say PB7=1.
7 Clocks and Counters

7.1 Overview
- A clock is a periodic signal used to synchronize events.
  - The CSU requires a clock to synchronize the execution of instructions.
  - Clocks are often used in conjunction with hardware counters.
- A hardware counter increments (counts) on the rising edge of the clock.
  - Counters have a limited range based on the number of bits in the counter.
  - When the counter goes from all ones to all zeros it is said to overflow.
  - Specific counters are connected to different clocks, resulting in different counting rates.
- The timer functions and other I/O features require clocks and counters to perform their operations.

7.2 Clocks
- EXTAL
  - Input to 68HC11. Supplied by external circuitry.
- XTAL
  - Output from the 68HC11 for use by external devices.
  - Highest internal clock frequency. Use for synchronization of instructions.
  - The stop disable bit is used to put the 68HC11 to sleep (turn off XTAL).
- E Clock
  - Internal 68HC11 clock.
  - Down-sampled XTAL clock by four (1/4 frequency).
  - Base clock for timer functions. A machine cycle, or tick, is equal to one period of the E clock.
  - Other clocks on 68HC11 based on E-clock.
- EVBU Board
  - External 8 MHz crystal oscillator
    - EXTAL = 8 MHz
    - XTAL = 8 MHz
    - E Clock = 2 MHz (1 tick = 0.5 µsec)
7.3 Free Running Clock and Counter

7.3.1 Overview

- The free running clock is based off of the E clock.
- Configuration bits PR1 and PR0 are timed write-once bits (you can not change them).
- The free running counter (TCNT) is a 16 bit I/O register.
- The timer functions are based on the free running clock/counter.

![Free Running Clock and Counter (TCNT)](image)

<table>
<thead>
<tr>
<th>PR1</th>
<th>PR0</th>
<th>Scale</th>
<th>Frequency (kHz)</th>
<th>Period (µs)</th>
<th>Overflow (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2000</td>
<td>0.5</td>
<td>32.768</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>500</td>
<td>2.0</td>
<td>131.072</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8</td>
<td>250</td>
<td>4.0</td>
<td>262.144</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>16</td>
<td>125</td>
<td>8.0</td>
<td>524.288</td>
</tr>
</tbody>
</table>

Free Running Counter I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>TCNT (Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TCNT (Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TMSK2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>PR1</td>
<td>PR0</td>
</tr>
</tbody>
</table>
7.3.2 Sample Source Code

tcnt.c (CMM11E1, CME11E9, simulator)
tcndelay.c (CMM11E1, CME11E9)
7.4 Real-Time Interrupt Clock

- The real-time clock is based off of the E clock.
- Configuration bits RTR1 and RTR0 can be modified.
- The period of the real-time clock is on the order of milliseconds, which corresponds to typical update rates for real-time applications.

![Diagram of E Clock, ↓ RTR1, RTR0 (PACTL), Real-Time Clock]

<table>
<thead>
<tr>
<th>RTR1</th>
<th>RTR0</th>
<th>Scale</th>
<th>Period (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$2^{13}$</td>
<td>4.096</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$2^{14}$</td>
<td>8.192</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$2^{15}$</td>
<td>16.384</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$2^{16}$</td>
<td>32.768</td>
</tr>
</tbody>
</table>

### Real-Time Interrupt (RTI) Clock

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>PACTL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>RTR1</td>
<td>RTR0</td>
</tr>
</tbody>
</table>
7.5 Software Timing Measurements

- The free running counter can be used to perform timing measurements.
- The accuracy of the timing measurements is limited by the period of the free running clock and the size of the free running counter.
  - Smallest measurable time interval is 1 clock period (0.5 \( \mu \text{sec} \)).
  - Largest measurable time interval is 1 counter cycle (32.768 msec).
  - Only relative time can be measured (i.e. second hand only).
- In addition, there are limitations and considerations that need to be taken into account depending on the method of measurement.
  - Software always requires time to execute.
  - All assembly instructions take at least 1 tick (0.5 \( \mu \text{sec} \)) to execute. Most assembly instructions take several ticks or more.
  - C statements typically correspond to several assembly instructions.
  - Hardware is capable of performing operations faster than software.
7.5.1 Example – Measuring a Pulse Width

- Objective
  - Measure the pulse width of the signal shown below

- Strategy
  - Connect signal to an input pin (pick PA2).
  - Wait for rising edge. Record time.
  - Wait for falling edge. Record time.
  - Take the difference to calculate pulse width.

- Sample Source Code
  tcntmpw.c (CME11E9, simulator)

```c
unsigned start, stop, width; /* Stop watch variables */
while ( (PORTA & PA2) == PA2); /* Ensure start during low part */
while ( (PORTA & PA2) == 0); /* Wait until PA2 high */
start = TCNT;       /* Record start time */
while ( (PORTA & PA2) == PA2); /* Wait until PA2 low */
stop = TCNT;        /* Record stop time */
width = stop - start; /* Calculate pulse width(ticks) */
```
• Observations
  o Time is always positive (unsigned), relative, and in ticks.
  o Delay between detection of edge and recording of time.

  ℋWindows1\Actual duration
  Measured duration

  Record start
  Record stop

  o Unsigned math can handle one counter overflow.
  o Does not detect wrap around times.

  Start = E000
  Stop = 1000
  Start = 1000
  Stop = 4000

  Start = A000
  Stop = D000

  Actual pulse width could be 3000, 13000, 23000, …

• Questions?
  o What is the minimum pulse width that can be measured using this method?
  o What is the maximum pulse width that can be measured using this method?
7.5.2 Example – Generating a Pulse Width

- Objective
  - Generate the signal shown below.

- Strategy
  - Connect signal to an output pin (pick PA6).
  - Set signal high.
  - Record time.
  - Wait for pulse width duration to elapse.
  - Set signal low.

- Sample Source Code
  tcntgpw.c (CME11E9, simulator)

```c
unsigned start, width; /* Stop watch variables */
width = 100; /* 50 usec pulse width */
PORTA |= PA6; /* Set PA6 high */
start = TCNT; /* Record start time */
while ( (TCNT-start) < width); /* Wait */
PORTA &= ~PA6; /* Set PA6 low */
```
- **Observations**
  - Delay between setting signal high and recording of start time.
  - Delay between end of while loop and setting signal low.
  - Cannot use equality comparisons.
  - Must use proper conditional statement in order to handle counter overflow.

```latex
\begin{tabular}{|c|c|c|c|}
\hline
TCNT & duration = TCNT - start & TCNT < stop & duration < width \\
\hline
4000 & 2000 & True & True \\
6000 & 4000 & False & False \\
1000 & F000 & True & False \\
\hline
\end{tabular}
```

- **Questions?**
  - What is the minimum pulse duration that can be generated using this method?
  - What is the maximum pulse duration that can be generated using this method?
8 Events and Flags

8.1 Overview

- How does software and hardware “communicate” with each other?
  - Software “communicates” with hardware by modifying I/O registers.
  - Hardware “communicates” with software by modifying I/O registers.
    - Software must check I/O registers to “notice” information.
    - Hardware sets a flag bit to indicate an event occurred.
- An event occurs whenever specific conditions are met.
  - Every event has its own set of conditions.
    - Alarm clock event occurs whenever the actual time matches the preset alarm time.
    - Light turns on when switch is thrown.
  - Event conditions are checked by hardware.
    - Conditions can be checked “instantaneously”.
    - It does not take up processor time.
- A flag is a bit located in an I/O register that indicates whether or not an event has occurred.
  - The purpose of a flag bit is to inform “software” that an event has occurred.
  - It is the responsibility of the “software” to check the flag bit.
  - Every event has its own flag bit.
  - A flag can only be set to a 1 by hardware. Software cannot set a flag.
  - Most flags can be cleared (set to a 0) by software by writing a 1 to the flag bit.
  - Flags are cleared on RESET.
  - Flag bits are denoted by <event name>F (i.e. the RTI flag is RTIF).
- There are two methods for responding to events: polling technique and interrupt technique.
  - Polling technique – program checks flags to determine if event occurs
  - Interrupt technique – hardware generates an interrupt whenever event occurs
8.2 Polling Technique

8.2.1 Overview

- Software periodically checks to see if a flag is set.
  - Good – Program controls when the flag is checked.
  - Bad – Can be a significant delay between event and when software checks the flag.

![Flow Chart for Polling Technique]

Figure 7: Flow Chart for Polling Technique

8.2.2 Sample Source Code

tofdelay.c (CMM11E1, CME11E9)
tof.c (CMM11E1, CME11E9, simulator)
8.3 Interrupt Technique

8.3.1 Overview

- Hardware will interrupt the execution of the program when an event occurs.
- An interrupt service routine (ISR) associated with the event will automatically be executed.
  - ISRs differ from normal subroutines in how they are called and how they return.
  - An ISR cannot be called directly in the program.
- When the ISR is completed, the program will resume from where it was interrupted.
  - Good – No need to waste time checking flags.
  - Good – More robust/structured programming.
  - Not so Bad – There is still a slight delay in software response.
  - Bad – Too many interrupts cause the program to starve.

![Flow Chart for Interrupt Technique]

Figure 8: Flow Chart for Interrupt Technique

8.3.2 Sample Source Code

`toi.c (CMM11E1, CME11E9, simulator)`
8.3.3 Terminology

- Synchronous / Asynchronous
  - Synchronous interrupts – current instruction is completed prior to executing ISR. This is a recoverable interrupt.
  - Asynchronous interrupts – current instruction is aborted and ISR is executed immediately. This is a non-recoverable interrupt.

- Maskable / Non-Maskable interrupts
  - Maskable interrupts – the interrupt can be disabled (masked).
  - Non-Maskable interrupts – the interrupt cannot be disabled (masked).
    - The I bit in the CCR register is used to enable (0)/disable (1) “all” maskable interrupt
    - The X bit in the CCR register is used to enable (0)/disable (1) the XIRQ interrupt
      - Once enabled the XIRQ cannot be disable except by a RESET.
      - There are additional individual enable bits for each maskable interrupt (and some “non-maskable” interrupts).

- Priority
  - Determines the order in which multiple events/interrupts that occur at the same time will be handled.

- Vectors
  - A vector is an address that corresponds to a subroutine (points to a subroutine)
  - An interrupt vector (IV) contains the address of an ISR.
  - An interrupt vector table (IVT) contains a list of interrupt vectors. Each vector in the table corresponds to a specific interrupt event. The IVT is located in ROM.
  - A pseudo-interrupt vector (PIV) contains a jump statement to an ISR.
  - A pseudo-interrupt vector table (PIVT) contains a list of pseudo-interrupt vectors. The PIVT is located in RAM to allow assignment of ISRs during the development process.
  - The IVT is mapped to the PIVT on the EVBU.

![Pseudo-Vector Table](image)

| $00C4 | ... |
| $00F4 | 7E |
| $00F5 | ?? |
| $00F6 | ?? |
| ... |
| $00FF |

![Vector Table](image)

| $FFC0 | ... |
| $FFFF | 00 |
| $FFFF | F4 |
| ... |
| $FFFF |

Start of SWI ISR (JMP xxxx)
Extended address of SWI ISR (00F4)

Figure 9: Interrupt Vector Example
<table>
<thead>
<tr>
<th>Vector Address (2 bytes)</th>
<th>Pseudo-Vector Address (3 bytes)</th>
<th>Source in order of priority (low to high)</th>
<th>CCR Mask Bit</th>
<th>Local Mask Bit</th>
<th>Synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFD6 00C4</td>
<td></td>
<td>Serial Communication Interface (SCI)</td>
<td>I</td>
<td>RIE TIE TCIE ILIE</td>
<td>Y</td>
</tr>
<tr>
<td>FFD8 00C7</td>
<td></td>
<td>Serial Peripheral Interface (SPI)</td>
<td>I</td>
<td>SPIE</td>
<td>Y</td>
</tr>
<tr>
<td>FFDA 00CA</td>
<td></td>
<td>Pulse Accumulator Input Edge</td>
<td>I</td>
<td>PAII</td>
<td>Y</td>
</tr>
<tr>
<td>FFDC 00CD</td>
<td></td>
<td>Pulse Accumulator Overflow</td>
<td>I</td>
<td>PAOVI</td>
<td>Y</td>
</tr>
<tr>
<td>FFDE 00D0</td>
<td></td>
<td>Timer Overflow</td>
<td>I</td>
<td>TOI</td>
<td>Y</td>
</tr>
<tr>
<td>FFE0 00D3</td>
<td></td>
<td>Timer Input Capture 4 / Timer Output Compare 5</td>
<td>I</td>
<td>I4/OC5I</td>
<td>Y</td>
</tr>
<tr>
<td>FFE2 00D6</td>
<td></td>
<td>Timer Output Compare 4</td>
<td>I</td>
<td>OC4I</td>
<td>Y</td>
</tr>
<tr>
<td>FFE4 00D9</td>
<td></td>
<td>Timer Output Compare 3</td>
<td>I</td>
<td>OC3I</td>
<td>Y</td>
</tr>
<tr>
<td>FFE6 00DC</td>
<td></td>
<td>Timer Output Compare 2</td>
<td>I</td>
<td>OC2I</td>
<td>Y</td>
</tr>
<tr>
<td>FFE8 00DF</td>
<td></td>
<td>Timer Output Compare 1</td>
<td>I</td>
<td>OC1I</td>
<td>Y</td>
</tr>
<tr>
<td>FFEA 00E2</td>
<td></td>
<td>Timer Input Capture 3</td>
<td>I</td>
<td>IC3I</td>
<td>Y</td>
</tr>
<tr>
<td>FFEB 00E3</td>
<td></td>
<td>Timer Input Capture 2</td>
<td>I</td>
<td>IC2I</td>
<td>Y</td>
</tr>
<tr>
<td>FFEC 00E5</td>
<td></td>
<td>Timer Input Capture 1</td>
<td>I</td>
<td>IC1I</td>
<td>Y</td>
</tr>
<tr>
<td>FFED 00E6</td>
<td></td>
<td>Real-Time Interrupt</td>
<td>I</td>
<td>RTI</td>
<td>Y</td>
</tr>
<tr>
<td>FFE2 00EE</td>
<td></td>
<td>IRQ</td>
<td>I</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>FFE4 00F1</td>
<td></td>
<td>XIRQ</td>
<td>X</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>FFE6 00F4</td>
<td></td>
<td>Software Interrupt (SWI)</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>FFE8 00F7</td>
<td></td>
<td>Illegal Opcode</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>FFFA 00FA</td>
<td></td>
<td>Computer Operating Properly (COP)</td>
<td></td>
<td></td>
<td>NOCOP</td>
</tr>
<tr>
<td>FFFC 00FD</td>
<td></td>
<td>Clock Monitor</td>
<td></td>
<td></td>
<td>CME</td>
</tr>
<tr>
<td>FFFE</td>
<td></td>
<td>RESET (Bootloader Start)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.3.4 Sequence of Operation

- When a synchronous interrupt occurs the HW sets an interrupt flag.
- When the current instruction cycle is completed, the HW processes the interrupt.
  - If multiple interrupts occurred, the interrupt with the highest priority is processed first. The other interrupts will remain in a queue to be processed later.
- The CPU registers are pushed onto the stack to be preserved.
- The I and X bits in the CCR register are set to prevent the ISR from being interrupted.
  - If interrupts occurs during an ISR they will be placed in the queue.
- The corresponding IV is fetched from the IVT and placed in the PC (ISR starts executing).
- The RTI instruction restores all the CPU registers to their pre-interrupted state.
9 Timer Overflow

9.1 Overview

- Event occurs when the free running counter overflows.
- Periodic event.
- No external pins associated with event.
- Operates at the same time scale as real time interrupt event.

### Timer Overflow Event

<table>
<thead>
<tr>
<th>Flag</th>
<th>Interrupt Enable</th>
<th>Control Registers</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOF</td>
<td>TOI</td>
<td>PR1, PR0</td>
<td>None</td>
</tr>
<tr>
<td>(TFLG2)</td>
<td>(TMSK2)</td>
<td>(TMSK2)</td>
<td></td>
</tr>
</tbody>
</table>

### Timer Overflow I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>TMSK2</td>
<td>TOI</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>PR1</td>
<td>PR0</td>
</tr>
<tr>
<td>TFLG2</td>
<td>TOF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

9.2 Sample Source Code

tocntr.c (CMM11E1, CME11E9, simulator)
10 Real-Time Interrupt

10.1 Overview

- Event occurs when the real time clock ticks.
- Periodic event.
- No external pins associated with event.

<table>
<thead>
<tr>
<th>Real-Time Interrupt Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
</tr>
<tr>
<td>RTIF (TFLG2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real-Time Interrupt I/O Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>TMSK2</td>
</tr>
<tr>
<td>TFLG2</td>
</tr>
<tr>
<td>PACTL</td>
</tr>
</tbody>
</table>

10.2 Sample Source Code

rticntr.c (CMM11E1, CME11E9, simulator)
11 Output Compares

11.1 Overview

- Event occurs when TCNT equals TOCx (Output Compares).
- Alarm clock type event.
- External pin(s) associated with each event.
- Output pin(s) can be changed automatically when event occurs (Output Compares).

![Diagram showing TCNT and TOCx relationships]

### Output Compare Events (1...5)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Interrupt</th>
<th>Control Registers</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC1F</td>
<td>OC1I</td>
<td>(OC1M, OC1D)</td>
<td>OC1...OC5</td>
</tr>
<tr>
<td>(TFLG1)</td>
<td>(TMSK1)</td>
<td>(TOC1)</td>
<td>PA7...PA3</td>
</tr>
<tr>
<td>OCxF</td>
<td>OCxI</td>
<td>OMx, OLx</td>
<td>OC2...OC5</td>
</tr>
<tr>
<td>(TFLG1)</td>
<td>(TMSK1)</td>
<td>(TCTL1)</td>
<td>PA6...PA3</td>
</tr>
</tbody>
</table>

### Output Compare I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>OC1M</td>
<td>OC1M7</td>
<td>OC1M6</td>
<td>OC1M5</td>
<td>OC1M4</td>
<td>OC1M3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OC1D</td>
<td>OC1D7</td>
<td>OC1D6</td>
<td>OC1D5</td>
<td>OC1D4</td>
<td>OC1D3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TOC1(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TOC1(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TOC2(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TOC2(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TOC3(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TOC3(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TOC4(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TOC4(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TI4O5(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TI4O5(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TCTL1</td>
<td>OM2</td>
<td>OL2</td>
<td>OM3</td>
<td>OL3</td>
<td>OM4</td>
<td>OL4</td>
<td>OM5</td>
<td>OL5</td>
</tr>
<tr>
<td>TMSK1</td>
<td>OC1I</td>
<td>OC2I</td>
<td>OC3I</td>
<td>OC4I</td>
<td>I4O5I</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TFLG1</td>
<td>OC1F</td>
<td>OC2F</td>
<td>OC3F</td>
<td>OC4F</td>
<td>I4O5F</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
11.2 Output Compare 1

- Event occurs when TCNT equals TOC1.
- Five external pins associated with event (OC1…OC5/PA7…PA3).
- Pins to control are selected by OC1M (OC1 mask register).
- The value of the controlled pins is determined by OC1D (OC1 data register).
- One event (OC1) can control five pins. All change at the same time.

### Output Compare 1 Control

| OC1My | 1 = Enables control of pin PAy  
| OC1M  | 0 = Disables control of pin PAy  
| OC1Dy | 1 = Set pin PAy high on OC1 event  
| OC1D  | 0 = Set pin PAy low on OC1 event  

### Output Compare 1 I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>OC1M</td>
<td>OC1M7</td>
<td>OC1M6</td>
<td>OC1M5</td>
<td>OC1M4</td>
<td>OC1M3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OC1D</td>
<td>OC1D7</td>
<td>OC1D6</td>
<td>OC1D5</td>
<td>OC1D4</td>
<td>OC1D3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

11.3 Output Compares 2…5

- Event occurs when TCNT equals TOCx.
- One external pin associated with each event (OC2/PA6, OC3/PA5, OC4/PA4, OC5/PA3).
- Control and value determined by associated OM and OL bits.
- Each event can control one pin.

### Output Compare 2…5 Control

<table>
<thead>
<tr>
<th>OMx</th>
<th>OLx</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disconnected (not controlled)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Toggle</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Low</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>High</td>
</tr>
</tbody>
</table>

### Output Compare 2…5 I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>TCTL1</td>
<td>OM2</td>
<td>OL2</td>
<td>OM3</td>
<td>OL3</td>
<td>OM4</td>
<td>OL4</td>
<td>OM5</td>
<td>OL5</td>
</tr>
</tbody>
</table>
11.4 Examples

11.4.1 Generating a Pulse Width (Output Compare 1)

- **Objective**
  - Generate the signal shown below.

- **Strategy**
  - Use Output Compare 1 event.
  - Connect signal to an output pin controlled by OC1 (pick OC1/PA6).
  - **Algorithm**
    - Configure OC1/PA6 to go high on next event.
    - Clear OC1 flag.
    - Wait for OC1 event to occur.
    - Configure OC1/PA6 to go low on next event.
    - Set new OC1 event time.
    - Clear OC1 flag.
    - Wait for OC1 event to occur.
    - Disable OC1 control.

- **Sample Source Code**
  - ocpw1.c (CME11E9, simulator)

- **Observations**
  - No delay between event time and signal change.
  - Limited only by time required by software to establish the next event time.

- **Questions?**
  - What is the minimum pulse duration that can be generated using this method?
  - What is the maximum pulse duration that can be generated using this method?
11.4.2 Generating a Pulse Width (Output Compare 2)

- **Objective**
  - Generate the signal shown below.

  ![Pulse Width Diagram]

- **Strategy**
  - Use Output Compare 2 event.
  - Connect signal to an output pin controlled by OC2 (OC2/PA6).
  - **Algorithm**
    - Configure OC2/PA6 to toggle on event.
    - Set PA6 low.
    - Clear OC2 flag.
    - Wait for OC2 event to occur.
    - Set new OC2 event time.
    - Clear OC2 flag.
    - Wait for OC2 event to occur.
    - Disable OC2 control.

- **Sample Source Code**
  - ocpw2.c (CME11E9, simulator)

- **Observations**
  - No delay between event time and signal change.
  - Limited only by time required by software to establish the next event time (shorter than previous example).
11.4.3 Generating a Pulse Width (Output Compare 1 and 2)

- Objective
  - Generate the signal shown below.

- Strategy
  - Use Output Compare 1 and 2 events.
  - Connect signal to an output pin controlled by OC1 and OC2 (OC2/PA6).
  - Algorithm
    - Configure OC1 to set signal high.
    - Configure OC2 to set signal low.
    - Set OC1 and OC2 times.
    - Clear OC1 and OC2 flags.
    - Wait for OC2 event to occur.
    - Disable OC1 and OC2 control.

- Sample Source Code
  - ocpwl2.c (CME11E9, simulator)

- Observations
  - No delay between event time and signal change.
  - No limitation due to software. Only limitation is due to hardware.
11.4.4 Generating a PWM Signal (Output Compare 1 and 2)

- Objective
  o Generate the periodic signal shown below.
  o By changing (modulating) the pulse width, the average voltage can be controlled.
  o Pulse Width modulation (PWM) is a method for generating an effective analog voltage from a digital signal.

- DutyCycle = \( \frac{TON}{TPERIOD} \)
  \( V_{ave} = 5 \cdot DutyCycle \)

- Strategy
  o Use Output Compare 1 and 2 events.
  o Connect signal to an output pin controlled by OC1 and OC2 (OC2/PA6).
  o Algorithm
    - Configure OC1 to set signal high.
    - Configure OC2 to set signal low.
    - Set OC1 time.
    - Begin loop
      - Clear OC1 and OC2 flags.
      - Wait for OC1 event to occur.
      - Set OC2 and next OC1 times.
    - Repeat loop

- Sample Source Code
  ocpwm.c (CME11E9, simulator)

- Observations
  o No delay between event time and signal change.
  o Software must have sufficient time after OC1 event to update OC2 time (limitation on minimum on time).

- Questions?
  o What is the minimum period/pulse width that can be generated using this method?
  o What is the maximum period/pulse width that can be generated using this method?
12 Input Captures

12.1 Overview

- One external pin associated with each event.
- Event occurs when corresponding input signal generates specified edge (Input Capture).
- When event occurs, TCNT is stored in the corresponding TICx register (Input Capture).
- Stopwatch type event.

![ICx Diagram](Image)

\[ \text{TICx} = \text{TCNT} \]

### Input Capture Events (1…4)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Interrupt Enable</th>
<th>Control Registers</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICxF</td>
<td>ICxI (TFLG1)</td>
<td>EDGxB,EDGxA (TCTL2)</td>
<td>IC1…IC3,IC4</td>
</tr>
<tr>
<td></td>
<td>(TMSK1)</td>
<td></td>
<td>PA2…PA0,PA3</td>
</tr>
</tbody>
</table>

### Input Capture Control

<table>
<thead>
<tr>
<th>Edge</th>
<th>EDGxB</th>
<th>EDGxA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disabled (not controlled)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Rising</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Falling</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Any</td>
</tr>
</tbody>
</table>

### Input Capture I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>TIC1(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TIC1(Lo)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TIC2(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TIC2(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TIC3(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TIC3(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TI40S(Hi)</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TI40S(Lo)</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TCTL2</td>
<td>EDG4B</td>
<td>EDG4A</td>
<td>EDG1B</td>
<td>EDG1A</td>
<td>EDG2B</td>
<td>EDG2A</td>
<td>EDG3B</td>
<td>EDG3A</td>
</tr>
<tr>
<td>TFLG1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1405F</td>
<td>IC1F</td>
<td>IC2F</td>
</tr>
<tr>
<td>ICx</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
12.2 Example – Measuring a Pulse Width

- **Objective**
  - Measure the pulse width of the signal shown below

  ![Pulse Width Diagram](image)

- **Strategy**
  - Connect signal to an input capture pin (pick IC1/PA2).
  - Configure IC1/PA2 for rising edge.
  - Clear IC1 flag. Wait for IC1 event to occur.
  - Record start time.
  - Configure IC1/PA2 for falling edge.
  - Clear IC1 flag. Wait for IC1 event to occur.
  - Record stop time.
  - Calculate pulse width (ticks)
  - Disable IC1 control.

- **Sample Source Code**
  - icpw.c (CME11E9, simulator)

- **Observations**
  - No delay between detection of edge and recording of time.
  - Limited only by time required by software to prepare for the next event time.

  ![Time to setup next TIC1 Diagram](image)

- **Questions?**
  - What is the minimum pulse width that can be measured using this method?
  - What is the maximum pulse width that can be measured using this method?
13 Analog-to-Digital Conversion

13.1 Overview

- An analog-to-digital converter (ADC) takes an analog signal (continuous value and time), samples it (discrete time), and converts the value into a digital (discrete value) representation.

- The voltage conversion range is determined by the $V_{rh}$ and $V_{rl}$ signals. On the EVBU, these signals are connected to +5 volts and 0 volts, respectively. The full-scale voltage range ($V_{FS}$) is the difference between the high and low reference voltages.
  
  \[
  V_{FS} = \left( V_{rh} - V_{rl} \right) = 5 \text{ volts}
  \]

- The number of bits on the ADC determines how accurately the analog voltage can be represented. The 68HC11 has one 8-bit ADC. The step-size/resolution is the amount of voltage range associated with one count.
  
  \[
  \Delta V = \left( \frac{V_{FS}}{2^N} \right) = \left( \frac{5}{256} \right) = 0.01953125 \text{ volts/count}
  \]

- The relationship between the analog value ($V_{in}$) and digital conversion ($D_{adc}$) is shown below.
  
  Digital conversion value: 
  \[
  D_{adc} = \left\lfloor \frac{V_{in} - V_{rl}}{\Delta V} \right\rfloor = \left\lfloor \frac{V_{in}}{\Delta V} \right\rfloor \text{ counts}
  \]

  Approximate analog value: 
  \[
  V_{in} \approx D_{adc} \cdot \Delta V \text{ volts}
  \]

- Example:
  
  \[
  V_{in} = 3.2 \text{ volts} \quad D_{adc} = \left\lfloor \frac{3.2}{0.01953} \right\rfloor = \left\lfloor 163.84 \right\rfloor = 163 \text{ counts}
  \]
  
  \[
  D_{adc} = 163 \text{ counts} \quad V_{in} \approx 163 \cdot 0.01953 = 3.1836 \text{ volts}
  \]
### 13.2 Configuration

#### Analog-to-Digital Conversion Control

<table>
<thead>
<tr>
<th>CD</th>
<th>CC</th>
<th>CB</th>
<th>CA</th>
<th>MULT=0</th>
<th>MULT=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>AN0→ADRx</td>
<td>AN0→ADR1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>AN1→ADRx</td>
<td>AN1→ADR2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>AN2→ADRx</td>
<td>AN2→ADR3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>AN3→ADRx</td>
<td>AN3→ADR4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>AN4→ADRx</td>
<td>AN4→ADR1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>AN5→ADRx</td>
<td>AN5→ADR2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>AN6→ADRx</td>
<td>AN6→ADR3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>AN7→ADRx</td>
<td>AN7→ADR4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>test signals</td>
<td>test signals</td>
</tr>
</tbody>
</table>

(You can ignore the CB CA bits)

#### Analog-to-Digital Conversion I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>ADCTL</td>
<td>CCF</td>
<td>0</td>
<td>SCAN</td>
<td>MULT</td>
<td>CD</td>
<td>CC</td>
<td>CB</td>
<td>CA</td>
</tr>
<tr>
<td>ADR1</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ADR2</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ADR3</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ADR4</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>OPTION</td>
<td>ADPU</td>
<td>CSEL</td>
<td>IRQE</td>
<td>DLY</td>
<td>CME</td>
<td>0</td>
<td>CR1</td>
<td>CR0</td>
</tr>
</tbody>
</table>

- The ADC requires extra power to operate. To conserve power, the ADC on the 68HC11 is initially turned off. The ADPU bit turns on/off the analog-to-digital converter (0=off, 1=on).
- There is one 8-bit analog-to-digital converter. A 16 input multiplexer is used to select which signal to convert from 16 possible sources. Eight sources are from Port E. The other eight are for testing purposes.
- The conversion process is started by writing to the ADCTL register. One conversion takes 64 ticks (32 usec). Conversions are always done in sets of four. The conversion results are stored in the four ADRx registers.
- SCAN bit determines number of conversion sets (0=one set, 1=continuous)
- MULT bit determines number of signals to convert per set (0=one signal, 1=four signals)
- CD, CC, CB, CA bits determine which signal(s).
- CCF flag is set by hardware when an entire set of conversions is completed. The flag bit is cleared by writing to the ADCTL register.
14 Serial Communication Interface (SCI)

14.1 Overview

- Serial communication is a standard interface used to transmit data one bit at a time across a single line.
- Because only one line is used to send the data, each bit is sent for a fixed time duration before the next bit is sent. On the receiver end, the line is checked at fixed time intervals to determine the bit value. The rate at which the bits are sent (line is checked) is called the BAUD rate. For example, a BAUD rate of 9600 means 9600 bits are sent every second.
- Bits are grouped into frames. Each frame has additional control bits (start, stop) for synchronization.

![Diagram showing serial communication process](image-url)
14.2 Baud Rate

**Serial Communication Interface (SCI) Clock**

<table>
<thead>
<tr>
<th>SCP1</th>
<th>SCP0</th>
<th>↓Scale</th>
<th>Prescaler Rate [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1*16</td>
<td>125000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3*16</td>
<td>41667</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4*16</td>
<td>31250</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>13*16</td>
<td>9600</td>
</tr>
</tbody>
</table>

**Serial Communication I/O Registers**

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>BAUD</td>
<td>-</td>
<td>-</td>
<td>SCP1</td>
<td>SCP0</td>
<td>-</td>
<td>SCR2</td>
<td>SCR1</td>
<td>SCR0</td>
</tr>
<tr>
<td>SCCR1</td>
<td>R8</td>
<td>T8</td>
<td>-</td>
<td>M</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SCCR2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>TE</td>
<td>RE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SCSR</td>
<td>TDRE</td>
<td>-</td>
<td>RDRF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SCDR</td>
<td>R7/T7</td>
<td>R6/T6</td>
<td>R5/T5</td>
<td>R4/T4</td>
<td>R3/T3</td>
<td>R2/T2</td>
<td>R1/T1</td>
<td>R0/T0</td>
</tr>
</tbody>
</table>
14.3 Transmit/Receive Data

**Serial Communication Interface (SPI)**

<table>
<thead>
<tr>
<th></th>
<th>Enable</th>
<th>Flag</th>
<th>Data</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit</td>
<td>TE</td>
<td>TDRE</td>
<td>M (SCCR1)</td>
<td>TxD (PD1)</td>
</tr>
<tr>
<td></td>
<td>(SCCR2)</td>
<td>(SCSR)</td>
<td>(SCDR)</td>
<td></td>
</tr>
<tr>
<td>Receive</td>
<td>RE</td>
<td>RDRF</td>
<td>M (SCCR1)</td>
<td>RxD (PD0)</td>
</tr>
<tr>
<td></td>
<td>(SCCR2)</td>
<td>(SCSR)</td>
<td>(SCDR)</td>
<td></td>
</tr>
</tbody>
</table>

- The Transmit Enable (TE) bit must be set to a 1 to enable transmission of serial data.
- The Receive Enable (RE) bit must be set to a 1 to enable reception of serial data.
- The Mode (M) bit selects the character format
  - 0 = start bit, 8 data bits, 1 stop bit
  - 1 = start bit, 9 data bits, 1 stop bit
- The Serial Communication Data Register (SCDR) is used for both the transmit data register (write only) and the receive data register (read only).
- Transmit Data Register Empty (TDRE) flag set to a 1 by hardware when the SCDR register is empty (ready to send new data). Flag cleared by reading the SCSR register and then writing to the SCDR.
- Receive Data Register Full (RDRF) flag set to a 1 by hardware when the SCDR register is full (new data received). Flag cleared by reading the SCSR register and then reading from the SCDR.
15 Pulse Accumulator

15.1 Overview

- Accumulates (counts) number of pulses (edges).
- Two modes of operation:
  - Event counting – asynchronous counting (counts pulses)
  - Gated time counting – synchronous counting (counts time)

15.2 Configuration

### Pulse Accumulator Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Flag</th>
<th>Interrupt Enable</th>
<th>Control Registers</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Edge</td>
<td>PAIF</td>
<td>PAII</td>
<td>(PACTL)</td>
<td>PA7</td>
</tr>
<tr>
<td></td>
<td>(TFLG2)</td>
<td>(TMSK2)</td>
<td>(PACNT)</td>
<td></td>
</tr>
<tr>
<td>Overflow</td>
<td>PAOVF</td>
<td>PAOVI</td>
<td>(PACTL)</td>
<td>PA7</td>
</tr>
<tr>
<td></td>
<td>(TFLG2)</td>
<td>(TMSK2)</td>
<td>(PACNT)</td>
<td></td>
</tr>
</tbody>
</table>

### Pulse Accumulator Control Register (PACTL)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAEN</td>
<td>Pulse accumulator enable</td>
<td>0</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>enable</td>
</tr>
<tr>
<td>PAMOD</td>
<td>Pulse accumulator mode</td>
<td>0</td>
<td>event counting mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>gated time accumulation mode</td>
</tr>
<tr>
<td>PEDGE</td>
<td>Pulse accumulator edge</td>
<td>0</td>
<td>falling edge (counting mode)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>rising edge (counting mode)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>active high (gated-time mode)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>active low (gated-time mode)</td>
</tr>
</tbody>
</table>

### Pulse Accumulator I/O Registers

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>PACTL</td>
<td>DDRA7</td>
<td>PAEN</td>
<td>PAMOD</td>
<td>PEDGE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PACNT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
15.3 Event Counting Mode

- PAEN = 1, PAMOD = 0
- PAI event occurs on specified edge of PA7 (PEDGE bit)
- PACNT register increments on PAI event
- PAOV event occurs on overflow of PACNT register

15.4 Gated Time Accumulation Mode

- PAEN = 1, PAMOD = 1
- PAI event occurs on de-assertion of PA7 (PEDGE bit)
- PACNT register increments while PA7 asserted
- PACNT register increments on pulse accumulator clock edge (E-Clock/64)
- PAOV event occurs on overflow of PACNT register
16 Interrupt Requests

- Occurs on specified signal level or edge

<table>
<thead>
<tr>
<th>Event</th>
<th>Flag</th>
<th>Interrupt Enable</th>
<th>Control Registers</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIRQ</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>XIRQ</td>
</tr>
<tr>
<td>IRQ</td>
<td>None</td>
<td>None</td>
<td>IRQE (OPTION)</td>
<td>IRQ</td>
</tr>
<tr>
<td>IRQ</td>
<td>STAF (PIOC)</td>
<td>STAI (PIOC)</td>
<td>EGA (PIOC)</td>
<td>STAA</td>
</tr>
</tbody>
</table>

![Interrupt Request (IRQ, XIRQ) Events Diagram](attachment:image-url)
Appendix A: Sample Source Code

standard.c (DOS)
/* FILENAME: standard.c  
* AUTHOR: A. Blauch, GVSU  
* COMPATIBILITY: DOS application 
* DESCRIPTION: 
* Sample source code for standard I/O functions 
* This will not compile with the gcc 68HC11 compiler 
* for the EVBU due to insufficient memory
*/
#include <stdio.h>

int main(void)
{
    int data=26;
    printf("Hello World\n");
    printf("sizeof(data) = %u\n",sizeof(data));
    printf("data = %d\n",data);
    printf("data = 0x%08X\n",data);
    return 0;
}
buffalo.c (CMM11E1, CME11E9, simulator)

/* FILENAME: buffalo.c */
/* AUTHOR: A. Blauch, GVSU */
/* COMPATIBILITY: CMM11E1, CME11E9, simulator */
/* DESCRIPTION: */
/* Sample source code for BUFFALO I/O functions */
/* First part identical to standard I/O example */

#include <buffalo.h>

int main(void)
{
    int data=26;
    char value;
    int counter;

    /* Same output as standard.c */
    puts("Hello World\n");
    puts("sizeof(data) = ");
    putint(sizeof(data));
    puts("\n");
    puts("data = ");
    putint(data);
    puts("\n");
    puts("data = 0x");
    puthex16(data);
    puts("\n");

    /* Example of BUFFALO I/O functions */

    /* Read and write characters */
    puts("Please press a key: ");
    value = getch();
    puts("\n");
    puts("You typed ");
    putch(value);
    puts("\n");

    /* Loops until character received */
    puts("Press any key to stop...");
    counter = 0;
    do {
        if (++counter == 1000) {
            counter = 0;
            puts(".");
        }
        value = input();
    } while (value==0);
    puts("You typed ");
    putch(value);
    puts("\n");

    /* Read and write integers */
    puts("Please enter a number: ");
    counter = getint();
    puts("\nYou typed ");
    putint(counter);
    puts("\n");
    return 0;
}
**int.c (CMM11E1, CME11E9, simulator)**

/* FILENAME: int.c  
* AUTHOR: A. Blauch, GVSU  
* COMPATIBILITY: CMM11E1, CME11E9, simulator  
* DESCRIPTION:  
* Sample source code for integer/floating-point math comparison  
*/

void func(int j)
{
}

int main(void)
{
    int j;
    for (j=0; j<100; j++)
    {
        func(j);
    }
    return 0;
}

**float.c (CMM11E1, CME11E9, simulator)**

/* FILENAME: float.c  
* AUTHOR: A. Blauch, GVSU  
* COMPATIBILITY: CMM11E1, CME11E9, simulator  
* DESCRIPTION:  
* Sample source code for integer/floating-point math comparison  
*/

void func(float j)
{
}

int main(void)
{
    float j;
    for (j=0; j<100; j++)
    {
        func(j);
    }
    return 0;
}
tcnt.c (CMM11E1, CME11E9, simulator)

/* FILENAME: tcnt.c
 * AUTHOR: A. Blauch, GVSU
 * COMPATIBILITY: CMM11E1, CME11E9, simulator (see comments)
 * DESCRIPTION:
 * Sample source code for free running counter
 * Used to measure how long instructions take to execute
 */

#include <buffalo.h>
#include <hc11e9.h>

void Wait(void) {
    puts("...Press any key..."); while ( input() == 0 ); puts("\n");
}

void DisplayTime(char *text, unsigned time)
{
    puts(text); putuint(time); puts(" ticks\n");
}

int main(void)
{
    unsigned start, stop;
    puts("File: "); puts(__FILE__); puts("\n");
    puts("TCNT timing demonstration.\n");
    getch();

    /* Continually display time */
    do {
        DisplayTime("TCNT = ",TCNT);
    } while ( input() == 0 );
    Wait();

    /* Measure how long it takes to update an I/O register */
    start = TCNT;
    PORTA = 0x23;
    stop = TCNT;
    DisplayTime("Start = ",start);
    DisplayTime("Stop = ",stop);
    DisplayTime("Time = ",stop-start);
    Wait();

    /* Measure how long it takes to transfer strings across serial port */
    /* !!! Timing does not behave properly on simulator !!! */
    start = TCNT;
    puts("1");
    stop = TCNT;
    DisplayTime("\nTime = ",stop-start);

    start = TCNT;
    puts("1234567890");
    stop = TCNT;
    DisplayTime("\nTime = ",stop-start);

    start = TCNT;
    puts("123456789012345678901234567890");
    stop = TCNT;
    DisplayTime("\nTime = ",stop-start);

    start = TCNT;
    puts("12345678901234567890123456789012345");
    stop = TCNT;
    DisplayTime("\nTime = ",stop-start);

    return 0;
}
tcntdelay.c (CMM11E1, CME11E9)

/* FILENAME: tcntdelay.c
 * AUTHOR: A. Blauch, GVSU
 * COMPATIBILITY: CMM11E1, CME11E9
 * DESCRIPTION:
 * Sample source code for free running counter
 * Used to perform a 5 second delay
 */

#include <buffalo.h>
#include <hc11e9.h>

/* This function uses a polling loop to check
 * the free running counter. It will wait
 * approximately the number of passed ticks
 * before returning.
 */
void delay(unsigned ticks)
{
    unsigned start;
    start=TCNT; /* Record starting time */
    /* Calculate elapsed time and compare */
    while ((TCNT-start)<ticks);
}

int main(void)
{
    int i;

    puts("File: "); puts(__FILE__); puts("\n");
    puts("TCNT 5 second delay.\n");
    /* Loop 5000 times - total delay of 5 second */
    for (i=0; i<5000; i++) {
        /* Delay approximately 1 msec */
        delay(2000);
    }
    return 0;
}
# tcntmpw.c (CME11E9, simulator)

/* FILENAME: tcntmpw.c
 * AUTHOR: A. Blauch, GVSU
 * COMPATIBILITY: CME11E9, simulator (tcntmpw_pa2.sti)
 * DESCRIPTION:
 * Sample source code for free running counter
 * Used to measure pulse width (PA2)
 */

#include <buffalo.h>
#include <hc11e9.h>

/* This function measures a positive pulse width
 * on PA2 and returns the time in ticks.
 */
unsigned MeasurePulseWidth(void)
{
    unsigned start, stop, width; /* Stop watch variables */
    while ( (PORTA & PA2) == PA2); /* Ensure start during low part */
    while ( (PORTA & PA2) == 0); /* Wait until PA2 high */
    start = TCNT; /* Record start time */
    while ( (PORTA & PA2) == PA2); /* Wait until PA2 low */
    stop = TCNT; /* Record stop time */
    width = stop - start; /* Calculate pulse width (ticks) */

    return width;
}

int main(void)
{
    unsigned width;

    puts("File: "); puts(__FILE__); puts("\n");
    puts("Measuring pulse width on PA2...\n");

    width = MeasurePulseWidth();
    puts("Width = "); putuint(width); puts(" ticks\n");

    return 0;
}
tcntgpw.c (CME11E9, simulator)

/* FILENAME:  tcntgpw.c
 * AUTHOR:   A. Blauch, GVSU
 * COMPATIBILITY:  CME11E9, simulator
 * DESCRIPTION:
 * Sample source code for free running counter
 * Used to generate pulse width (PA6)
 */

#include <buffalo.h>
#include <hc11e9.h>

/* This function generates a positive pulse width
 * on PA6. The width is passed as a parameter in ticks.
 */
void GeneratePulseWidth(unsigned width)
{
    unsigned start;   /* Stop watch variables */
    PORTA |= PA6;   /* Set PA6 high */
    start = TCNT;   /* Record start time */
    while ( (TCNT-start) < width); /* Wait */
    PORTA &= ~PA6;   /* Set PA6 low */
}

int main(void)
{
    unsigned width = 100;

    puts("File: "); puts(__FILE__); puts("\n");
    puts("Generating "); putuint(width); puts(" tick pulse on PA6.\n");
    GeneratePulseWidth(width);
    return 0;
}
# tofdelay.c (CMM11E1, CME11E9)

/* FILENAME:  tofdelay.c
 * AUTHOR:   A. Blauch, GVSU
 * COMPATIBILITY:  CMM11E1, CME11E9
 * DESCRIPTION:
  * Sample source code for timer overflow event
  * Used to perform a 5 second delay
 */

#include <buffalo.h>
#include <hc11e9.h>

int main(void)
{
  int i;

  puts("File: "); puts(__FILE__); puts("\n");
  puts("Timer overflow 5 second delay.\n");

  /* The free running counter overflows every 32.768 msec. 
     If we wait for the TO event to occur 160 times it will 
     correspond to a delay of approximately 5.1 seconds. 
     The first TO event will not occur after 32.768 msec because 
     we do not know exactly what time we entered the loop. */
  for (i=0; i<160; i++) {
    TFLG2 = TOF; /* Clear flag */
    while ((TFLG2 & TOF) == 0);
  }

  return 0;
}
tof.c (CMM11E1, CME11E9, simulator)

/* FILENAME: tof.c 
* AUTHOR: A. Blauch, GVSU 
* COMPATIBILITY: CMM11E1, CME11E9, simulator 
* DESCRIPTION: 
* Sample source code for timer overflow event 
* Continuously displays free running counter 
* Displays "hit" whenever TO event is noticed 
*/

#include <buffalo.h>
#include <hc11e9.h>

/* Required for usable simulation */
void Delay(void)
{
    unsigned start = TCNT;
    while ( (TCNT-start) < 2000 );
}

int main(void)
{
    char key;

    puts("File: "); puts(__FILE__); puts("\n");
    puts("Timer overflow event hit display (polling loop implementation).\n");
    puts("Press 'q' to quit.\n");
    puts("Press any key to begin...\n");
    getch();

    /* Polling loop */
    do {
        key = toupper(input());

        /* Check if TO flag set */
        if (TFLG2 & TOF) {
            /* TO event occured */
            TFLG2 = TOF; /* Clear flag */
            puts("Hit\n");
        }

        /* Display free running counter */
        puthex16(TCNT); puts("\n");
    } while (key!='Q');

    return 0;
}
to1.c (CMM11E1, CME11E9, simulator)

/* FILENAME:   toi.c
 * AUTHOR:     A. Blauch, GVSU
 * COMPATIBILITY:  CMM11E1, CME11E9, simulator
 * DESCRIPTION:
 * Sample source code for timer overflow interrupt
 * Continuously displays free running counter
 * Displays "hit" whenever TO interrupt occurs
 */

#include <buffalo.h>
#include <hc11e9.h>

/* ISR function declaration syntax */
void TO_ISR(void) __attribute__((interrupt));

/* ISR function definition */
void TO_ISR(void)
{
    TFLG2 = TOF;  /* Clear flag */
    puts("Hit\n");
}

int main(void)
{
    char key;

    puts("File: "); puts(__FILE__); puts("\n");
    puts("Timer overflow event hit display (interrupt implementation).\n");
    puts("Press 'q' to quit.\n");
    puts("Press any key to begin...\n");
    getch();

    /* Setup ISR */
    __asm__ ("sei");    /* Mask interrupts */
    *(unsigned *)0x00D1 = (unsigned)TO_ISR; /* Assign TO PIV */
    TMSK2 |= TOI;    /* Enable TO interrupt */
    TFLG2 = TOF;    /* Clear flag */
    __asm__ ("cli");    /* Unmask interrupts */

    /* Polling loop */
    do {
        key = toupper(input());
        /* Display free running counter */
        puthex16(TCNT); puts("\n");
    } while (key!='Q');

    /* Cleanup ISR */
    __asm__ ("sei");    /* Mask interrupts */
    TMSK2 &= ~TOI;    /* Disable TO interrupt */

    return 0;
}
tocntr.c (CMM11E1, CME11E9, simulator)

/* FILENAME: tocntr.c
 * AUTHOR: A. Blauch, GVSU
 * COMPATIBILITY: CMM11E1, CME11E9, simulator
 * DESCRIPTION:
 * Sample source code for timer overflow event
 * Implements 32.768 msec counter
 */

#include <buffalo.h>
#include <hc11e9.h>

unsigned CheckEvent(void)
{
    /* TO event configured to occur every 32.768 msec */
    if (TFLG2 & TOF)
    {
        TFLG2 = TOF; /* Clear flag */
        return 1;
    }
    return 0;
}

int main(void)
{
    char key;
    unsigned event_counter=0;
    unsigned polling_counter=0;

    puts("File: "); puts(__FILE__); puts("\n");
    puts("Timer overflow event counter (polling loop implementation).\n");
    puts("Press 'e' for event counter.\n");
    puts("Press 'p' for polling-loop counter.\n");
    puts("Press 'q' to quit.\n");

    do {
        event_counter += CheckEvent();
        polling_counter++;
        key = toupper(input());
        switch (key) {
            case 'E':
                putuint(event_counter); puts("\n");
                break;
            case 'P':
                putuint(polling_counter); puts("\n");
                break;
        }
    } while ( key != 'Q' );
    return 0;
}
rticntr.c (CMM11E1, CME11E9, simulator)

/* FILENAME: rticntr.c
 * AUTHOR: A. Blauch, GVSU
 * COMPATIBILITY: CMM11E1, CME11E9, simulator
 * DESCRIPTION:
 * Sample source code for real-time interrupt event
 * Implements 4.096 msec counter
 */

#include <buffalo.h>
#include <hc11e9.h>

unsigned CheckEvent(void)
{
    /* RTI event configured to occur every 4.096 msec */
    if (TFLG2 & RTIF)
    {
        TFLG2 = RTIF; /* Clear flag */
        return 1;
    }
    return 0;
}

int main(void)
{
    char key;
    unsigned event_counter=0;
    unsigned polling_counter=0;

    puts("File: "); puts(__FILE__); puts("\n");
    puts("Real-time interrupt event counter (polling loop implementation).\n");
    puts("Press 'e' for event counter.\n");
    puts("Press 'p' for polling-loop counter.\n");
    puts("Press 'q' to quit.\n");
    PACTL &= ~(RTR1 | RTR0); /* Set RTI to 4.096 msec */

    do {
        event_counter += CheckEvent();
        polling_counter++;
        key = toupper(input());
        switch (key) {
            case 'E':
                putuint(event_counter); puts("\n");
                break;
            case 'P':
                putuint(polling_counter); puts("\n");
                break;
        }
    } while ( key != 'Q' );

    PACTL &= ~(RTR1 | RTR0);
    return 0;
}
# File: ocpw1.c

## Description
Sample source code for output compares
Used to generate pulse width w/ OC1 (PA6)

```c
#include <buffalo.h>
#include <hc11e9.h>

/* This function generates a positive pulse width
 * on PA6. The width is passed as a parameter in ticks.
 */
void GeneratePulseWidth(unsigned width)
{
    TOC1 = TCNT + 200; /* Set to start in 100 usec. */
    OC1M = OC1M6; /* Configure OC1/PA7 to go high on next event. */
    OC1D = OC1D6;
    TFLG1 = OC1F; /* Clear OC1 flag. */
    while ((TFLG1 & OC1F) == 0); /* Wait for OC1 event to occur. */
    OC1D = 0; /* Configure OC1/PA7 to go low on next event. */
    TOC1 = TOC1 + width; /* Set new OC1 event time. */
    TFLG1 = OC1F; /* Clear OC1 flag. */
    while ((TFLG1 & OC1F) == 0); /* Wait for OC1 event to occur. */
    OC1M = 0; /* Disable OC1 control. */
}

int main(void)
{
    unsigned width = 100;
    puts("File: "); puts(__FILE__); puts("\n");
    puts("Generating "); putuint(width); puts(" tick pulse on PA6.\n");
    GeneratePulseWidth(width);
    return 0;
}
```
**ocpw2.c (CME11E9, simulator)**

/* FILENAME: ocpw1.c
* AUTHOR: A. Blauch, GVSU
* COMPATIBILITY: CME11E9, simulator
* DESCRIPTION:
* Sample source code for output compares
* Used to generate pulse width w/ OC2 (PA6)
*/

#include <buffalo.h>
#include <hc11e9.h>

/* This function generates a positive pulse width
* on PA6. The width is passed as a parameter in ticks.
*/
void GeneratePulseWidth(unsigned width)
{
    TOC2 = TCNT + 200; /* Set to start in 100 usec. */
    TCTL1 |= OL2; /* Configure OC2/PA6 to toggle on event. */
    TCTL1 &= ~OM2;
    PORTA &= ~PA6; /* Set PA6 low. */
    TFLG1 = OC2F; /* Clear OC2 flag. */
    while ((TFLG1 & OC2F) == 0); /* Wait for OC2 event to occur. */
    TOC2 = TOC2 + width; /* Set new OC2 event time. */
    TFLG1 = OC2F; /* Clear OC2 flag. */
    while ((TFLG1 & OC2F) == 0); /* Wait for OC2 event to occur. */
    TCTL1 &= ~(OM2 | OL2); /* Disable OC2 control. */
}

int main(void)
{
    unsigned width = 100;
    puts("File: "); puts(__FILE__); puts("\n");
    puts("Generating "); putuint(width); puts(" tick pulse on PA6.\n");
    GeneratePulseWidth(width);
    return 0;
}
ocpw12.c (CME11E9, simulator)

/* FILENAME: ocpw12.c */
/* AUTHOR: A. Blauch, GVSU */
/* COMPATIBILITY: CME11E9, simulator */
/* DESCRIPTION: */
/* Sample source code for output compares */
/* Used to generate pulse width w/ OC1 & OC2 (PA6) */
*/

#include <buffalo.h>
#include <hc11e9.h>

/* This function generates a positive pulse width */
/* on PA6. The width is passed as a parameter in ticks. */
void GeneratePulseWidth(unsigned width)
{
    TOC1 = TCNT + 200; /* Set to start in 100 usec. */
    OC1M = OC1M6; /* Configure OC1 to set signal high. */
    OC1D = OC1D6;
    TCTL1 |= OM2; /* Configure OC2 to set signal low. */
    TCTL1 &= ~OL2;
    TOC2 = TOC1 + width;
    TFLG1 = OC1F | OC2F; /* Clear OC1 and OC2 flags. */
    while ((TFLG1 & OC2F) == 0); /* Wait for OC2 event to occur. */
    OC1M = 0; /* Disable OC1 and OC2 control. */
    OC1D = 0;
    TCTL1 &= ~(OM2 | OL2);
}

int main(void)
{
    unsigned width = 100;
    puts("File: "); puts(__FILE__); puts("\n");
    puts("Generating "); putuint(width); puts(" tick pulse on PA6.\n");
    GeneratePulseWidth(width);
    return 0;
}
ocpwm.c (CME11E9, simulator)

/* FILENAME: ocpwm.c
 * AUTHOR: A. Blauch, GVSU
 * COMPATIBILITY: CME11E9, simulator
 * DESCRIPTION:
 * Sample source code for output compares
 * Used to generate pwm signal w/ OC1 & OC2 (PA6)
 */

#include <buffalo.h>
#include <hc11e9.h>

/* This function generates a pulse width modulated signal
 * on PA6. The width and period are passed as a parameter in ticks.
 */
void GeneratePWM(unsigned width, unsigned period)
{
    unsigned num_pulses = 10;
    TOC1 = TCNT + 200;       /* Set to start in 100 usec. */
    TOC2 = TCNT;
    OC1M = OC1M6;            /* Configure OC1 to set signal high. */
    OC1D = OC1D6;
    TCTL1 |= OM2;            /* Configure OC2 to set signal low. */
    TCTL1 &= ~OL2;
    do {
        TFLG1 = OC1F | OC2F;  /* Clear OC1 and OC2 flags. */
        while ((TFLG1 & OC1F) == 0); /* Wait for OC1 event to occur. */
        TOC2 = TOC1 + width;    /* Set OC2 and next OC1 times. */
        TOC1 = TOC1 + period;
    } while ( num_pulses-- );
    OC1M = 0;                /* Disable OC1 and OC2 control. */
    OC1D = 0;
    TCTL1 &= ~(OM2 | OL2);
}

int main(void)
{
    unsigned width = 100, period = 300;
    puts("File: "); puts(__FILE__); puts("\n");
    puts("Generating PWM on PA6.\n");
    putuint(width); puts(" tick on time\n");
    putuint(period); puts(" tick period\n");
    GeneratePWM(width, period);
    return 0;
}
icpw.c (CME11E9, simulator)

/* FILENAME: icpw.c 
* AUTHOR: A. Blauch, GVSU 
* COMPATIBILITY: CME11E9, simulator (icpw_pa2.sti) 
* DESCRIPTION: 
* Sample source code for input captures 
* Used to measure pulse width w/ IC1 (PA2) 
*/

#include <buffalo.h> 
#include <hc11e9.h>

/* This function measures a positive pulse width 
* on PA2 and returns the time in ticks. 
*/
unsigned MeasurePulseWidth(void) 
{ 
    unsigned start, stop, width; 
    TCTL2 |= EDG1A; /* Configure IC1/PA2 for rising edge. */ 
    TCTL2 &= ~EDG1B; 
    TFLG1 = IC1F; /* Clear IC1 flag. */ 
    while ((TFLG1 & IC1F) == 0); /* Wait for IC1 event to occur. */ 
    start = TIC1; /* Record start time */ 
    TCTL2 |= EDG1B; /* Configure IC1/PA2 for falling edge. */ 
    TCTL2 &= ~EDG1A; 
    TFLG1 = IC1F; /* Clear IC1 flag. */ 
    while ((TFLG1 & IC1F) == 0); /* Wait for IC1 event to occur. */ 
    stop = TIC1; /* Record stop time */ 
    width = stop - start; /* Calculate pulse width (ticks) */ 
    TCTL2 &= ~(EDG1A | EDG1B); /* Disable IC1 control. */ 
    return width; 
}

int main(void) 
{ 
    unsigned width; 
    puts("File: "); puts(__FILE__); puts("\n"); 
    puts("Measuring pulse width on PA2\n"); 
    width = MeasurePulseWidth(); 
    puts("Width = "); putuint(width); puts(" ticks\n"); 
    return 0;
}
Appendix B: Sample Programming Problems

1. Write a program that sets the output signal on PD2 equal to the complement of the input signal on PD3. The program should loop continuously until the input signal on PE3 becomes high.

2. Write a program that generates a digital output signal that is the complement of a digital input signal (software implementation of a NOT gate). The program should loop continuously. You may decide which pin(s) to use.

3. Write a program that displays on HyperTerminal the alphabet in reverse order. The letters are to be displayed at a rate of one character every 1 sec.

4. Write a program that generates a 10 kHz, 60% duty cycle signal. The program should loop continuously. You may decide which pin(s) to use.

5. Write a program that generates a single 0.5 µsec pulse. The program should generate one pulse and then exit. You may decide which pin(s) to use.

6. Write a program that will measure the period of a square wave (50% duty cycle) input signal. The period is to be displayed on HyperTerminal in µsec. The program should perform one measurement and then exit. You may decide which pin(s) to use.

7. Write a program to determine the time-shift between two digital input signals. The time-shift should be measured between the rising edges and displayed on HyperTerminal in µsec. The program should perform one measurement and then exit. You may decide which pin(s) to use.

8. Write a program that calculates the average (DC) value of an analog input signal. You must take a measurement every millisecond. The average must be calculated over a 100 msec interval and displayed on HyperTerminal in counts. The program should perform one measurement and then exit. You may decide which pin(s) to use.

9. Write a program to determine the turn-on threshold voltage for the 68HC11 digital input circuitry. The turn-on voltage is to be displayed on HyperTerminal in counts. The program should perform one measurement and then exit. You may decide which pin(s) to use.

10. Write a program that compares two analog input signals, $V_1$ and $V_2$. A digital output signal must be set to a 1 if $V_1 > V_2$, and to a 0 if $V_1 \leq V_2$ (software implementation of a comparator circuit). The program should loop continuously. You may decide which pin(s) to use.