68HC11 Operating Modes
Modes

• Single-Chip
• Expanded Multiplexed
• Special Bootstrap
• Special Test
Minimal Circuit Layout: Single Chip
Timing Diagrams
Timing

Figure 10-32. Timer Counter as MCU Leaves Reset

Figure 10-33. Timer Counter Read — Cycle-by-Cycle Analysis
Laboratory 2 Debrief
Exercise 1: Serial TX

• Generally well done

• Emphasis on the “Serial String Transmission” module.

• Use of Null or CR:LF for termination both fine (as long as you didn’t transmit the null).

• Bitwise Ops for Config

• If you’re doing something “Hacky”, DOCUMENT
Exercise 1: Pseudocode

Main:
1. Initialise Serial
2. TX String 1
3. Delay
4. TX String 2
5. Delay
6. Goto (2)

TX String:
1. Load Character
2. If Null, RETURN
3. Else, Poll on TDRE
4. Send Character
5. Increment Pointer
6. Goto (1)
Exercise 1: Sample Code (NULL)

<table>
<thead>
<tr>
<th>SerialTX</th>
<th>PollTDRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA 0, Y</td>
<td>PSHA</td>
</tr>
<tr>
<td>BEQ Done</td>
<td>LDDAA SCSR</td>
</tr>
<tr>
<td>JSR PollTDRE</td>
<td>ANDA #TDREMask</td>
</tr>
<tr>
<td>STAA SCDR</td>
<td>BEQ PollTDRE</td>
</tr>
<tr>
<td>INY</td>
<td>PULA</td>
</tr>
<tr>
<td>BRA SerialTX</td>
<td>RTS</td>
</tr>
<tr>
<td>Done: RTS</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 1: Sample Code (CR:LF)

SerialTX:
- LDAA 0,Y
- JSR PollTDRE
- STAA SCDR
- INY
- CMPA #LF
- BNE SerialTX
- RTS

PollTDRE:
- PSHA
- LDAA SCSR
- ANDA #TDREMask
- BEQ PollTDRE
- PULA
- RTS
**Exercise 1: Main Loop**

**InitSerialTX:**

- **LDAA** #SCCR1Cfg
- **STAA** SCCR1
- **LDAA** #SCCR2Cfg
- **STAA** SCCR2
- **LDAA** #BAUDCfg
- **STAA** BAUD
- RTS

**JSR InitSerialTX**

**MainLoop:**

- **LDY** #String1
- **JSR** SerialTX
- **JSR** Delay
- **LDY** #String2
- **JSR** SerialTX
- **JSR** Delay
- **BRA** MainLoop
Exercise 1e: Serial RX

• Solution Trivial
• Config from previous part with RX rather than TX

• REMEMBER TO POLL ON RDRF

• SCDR → PORTB
Exercise 2: Timers and Interrupts

• A lot of problems

• Good Design is Key

• Understand the links between Timers and Interrupts

• Understand Interrupt!
Exercise 2: Good Design Practice

• What on-chip systems am I using?
  – Timers (Timer Output Compare X)
  – Interrupts
  – DIP Switches + Maths

• Design each of these modules as independently as possible

• Glue them together
Exercise 2: Underlying Maths

- E-Clock Frequency = 2 MHz
- Max Required Timing Interval = 71 mSec

- = 142000 cycles to count
- More than $2^{16}$, so either prescale or count overflows.

- Make a JUSTIFIED decision
Prescaler

- Bits PR1 and PR0 set the value of the prescaler (TMSK2)

- The prescaler can have the value 1, 4, 8 or 16

- If the prescaler value is 1, every E-clock cycle increments the free running counter

- PR1 and PR0 must be set in the first 64 clock cycles

- We can do it with Init function from the evaluation board setting TMSK2 or modifying ther Parameter file in the loader
Exercise 2: Interrupt System

• Need to appropriately manage:
  1. Global Interrupt Mask
  2. Local Interrupt Mask
  3. Local Interrupt Flag

• (1) and (2) are configured at startup (InitOCX)
• (3) is managed by the timer subsystem, so we need to use that appropriately.
Sample Code: Maths

• Base Equations:

\[
T_H = \frac{DC}{DC_{\text{Max}}} \times T
\]
\[
T_L = T - T_H
\]

• What can we Pre-Compute?

• \( T / DC_{\text{Max}} \), then at runtime we only need multiply and subtract

• Pre-Calculate Fraction:
  – MSF = round(T/255)

• Multiply PORTC by MSF

• This architecture does not enable us to deal with rounding errors.

• Necessary???

• Why is \( DC_{\text{Max}} 255 \)?
Sample Code: Maths

• Pre-Calculate Fraction:
  – MSF = round(T/255)

• Multiply PORTC by MSF

• This architecture does not enable us to deal with rounding errors.

• Necessary???

T = 142000 / 4 = 35500
MSF = 35500/255 = 139.21568627

@ 255/255, t = 139*255
  = 35445

Error = 55 Cycles = 0.155%

Balance the error (offset)?
## Sample Code: Maths

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSF</td>
<td>EQU 139</td>
</tr>
<tr>
<td>Offset</td>
<td>EQU 27</td>
</tr>
<tr>
<td>Period</td>
<td>EQU 35500</td>
</tr>
</tbody>
</table>

**Code Goes Here**

```assembly
DoCalcs:
LDAA #MSF
LDAB PORTC
MUL
ADD #Offset
STD HiTime
LDD #Period
SUBD HiTime
STD LowTime
RTS
```

HiTime RMB 2
LowTime RMB 2
Pseudocode: Timer/ISR

**TIMER**

*Using TOC2 only*

- Select Drive Direction
- Initialise Compare Module to Drive Pin
- Un-Mask TOCX Interrupt
- Each time timer match occurs:
  - Toggle Drive Direction
  - Increment TOC2 by appropriate time

**ISR**

- Clear Interrupt Flag
- Toggle Drive Direction
- If Next Drive Low:
  - Increment by HiTime
- Else
  - Increment by LowTime
- Return
Sample Code: Timer/ISR

InitTOC2:
- LDAA #OC2I
- ORAA TMSK1
- STAA TMSK1
- LDAA #TCTL1Init
- STAA TCTL1
- LDD TCNT
- ADD LowTime
- STD TOC2

TOC2ISR:
- LDAA #TOC2F 2
- STAA TFLG1 4
- EORA #OL2 2
- ANDA #OL2 2
- BEQ TimeHi 3
- BNE TimeLo 3
Sample Code: ISR

TimeHi:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDD</td>
<td>HiTime</td>
<td>5</td>
</tr>
<tr>
<td>BRA</td>
<td>ISRFinish</td>
<td>3</td>
</tr>
</tbody>
</table>

ISRFinish:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD</td>
<td>TOC2</td>
<td>6</td>
</tr>
<tr>
<td>STD</td>
<td>TOC2</td>
<td>5</td>
</tr>
<tr>
<td>RTI</td>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>

TimeLo:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDD</td>
<td>LowTime</td>
<td>5</td>
</tr>
<tr>
<td>BRA</td>
<td>ISRFinish</td>
<td>3</td>
</tr>
</tbody>
</table>
Sample Code: Validation

- ISR will take 55 E-cycles to execute. (as long as DoCalcs is outside the ISR)
- Call to ISR will take ~14 E-Cycles (undocumented!)
- Shortest time we need is Offset * 4 E-cycles:
  \[= 27 \times 4 = 108 \text{ E-cycles}\]
- Therefore this module will meet spec (with appropriate caveats)

- Special cases for 100% and 0% duty cycle?
  - PORTC = 0 → 0% DC
  - PORTC = 255 → 100% DC
  - Known error proportional to DC:
    - +0.077% @ 0%DC
    - 0% @ 50%DC
    - -0.077% @ 100% DC
Sample Code: Stick it all Together

```
ORG $8800
CLRA
STAA DDRC
JSR DoCalcs
JSR InitTOC2
CLI
Main:
  JSR DoCalcs
BRA Main
```

- Lastly, need to load the address of TOC2ISR into the interrupt vector table.

```
% Ref Table 9-3
ORG $FFE6
FDB TOC2ISR
```
Laboratory 2: General Comments

• HASHES!!!!!
  – Addressing modes still causing problems!
  – Re-read the extra lecture from Week 5

• Interrupt Vector Table initialisation
  – In code (LDD #ISRName; STD VectorLocation)
  – In initialisation ( ORG VectorLocation; FDB ISRName )

• ISRs should be as quick as possible. The longer they are the more likely strange issues are to occur.
Laboratory 2: General Comments

• Use the design workflow we’ve been talking about all semester:
  Design → Modules → Pseudocode → Comments

• Then fill in the code required to perform the actions in the comments.

• Go back and correct comments later.
Final Comments

• Most of the problems seem to be symptomatic of poor design and lack of background
  – Need to do the preparation before going to the lab!
  – We expect you to go with an initial version of the program

• Try to start your pre-work early. It gives time to ask questions and get answers.