SERVOMOTOR CONTROLLER.

(LABORATORY III)

3/3/2004

Sergey Averchenkov

CS345, Queens College, N.Y.

INTRODUCTION:

The reason for this lab was to learn how to use Handel-C to interface with external devices attached to ATA/Expansion header on the RC-200E. We tried generating pulses of the specified width on the output pins on the RC-200E. The specification of pulse widths was provided by the servomotor used as an external device in this lab. (Pulses of various widths tell servomotor to rotate either clockwise of counterclockwise.) In addition, in the process of developing this application, we explored channels as the method for synchronization of several concurrent threads.

METHOD:

In order to test Handel-C interface with output devices, we first has to learn how to simulate the same behavior in software using Waveform Analyzer.

Worked Through the First Waveform Analyzer Example:

Created a DK Workspace named "Laboratory III" for this lab in "My Projects" directory. Added a new project named "View Waveforms" to this workspace, and added a Handel-C source file named *view_waveforms.hcc* to the project. We used default configuration for the project.

Consulted the Waveform Analyzer Manual (in \Celoxica\DK\Documentation directory) and followed the directions on pages 5-8 for tracing the execution of a simple Handel-C program.

(See Appendix A: Waveform Analyzer Example Source Code.)

Wrote a Program that Outputs to a Pin:

Created a second project named "Generate Pulses" in the Laboratory III workspace, and configured it for EDIF and Simulation. Wrote a macro procedure named microsec_delay() that delays the program for the number of microseconds passed as a parameter. Wrote a program that endlessly generates a 1 msec pulse every 20 msec to the output of the Pin #3 of the RC200E expansion header. We verified the program using waveform analyzer and oscilloscope.

(See Appendix B: Generate Pulses Source Code.)

Write a Program that Controls a Servomotor

Created a third project named "Motor Control" in the Laboratory III workspace, and configured it for EDIF and Simulation.

We wrote the code according to the following specifications:

(See Appendix C: Servomotor Source Code.)

Project Specifications:

Program is to generate a pulse every 20 msec. If one pushbutton is pressed, the pulse is to be 1.0 msec long; if the other button is pressed, the pulse is to be 2.0 msec long. If neither button (or both buttons) is pressed, the pulse is to be 1.5 msec long. The servomotors we

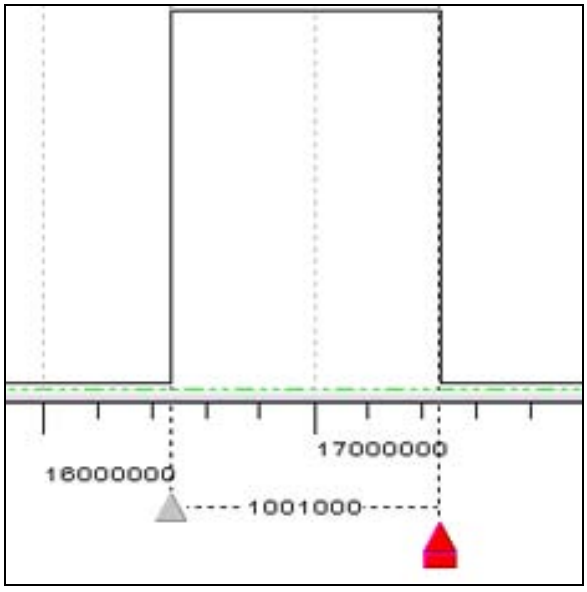
are using, Futaba model number 3003, rotates clockwise when it receives 1.0 msec pulses, counterclockwise when it receives 2.0 msec pulses, and not at all when it receives 1.5 msec pulses. (Clockwise and counterclockwise depend on your point of view.) Your program must use one endless loop to time the millisecond intervals and another endless loop to generate the pulses. The two loops must synchronize with each other using a Handel-C channel.

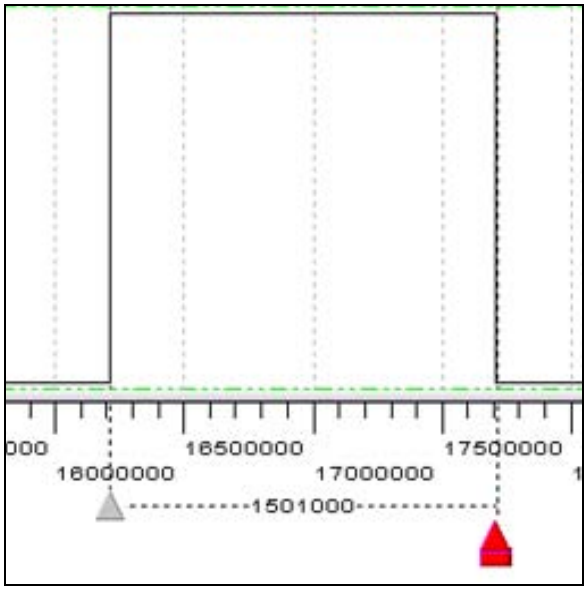
We verified the code using both the oscilloscope and waveform analyzer.

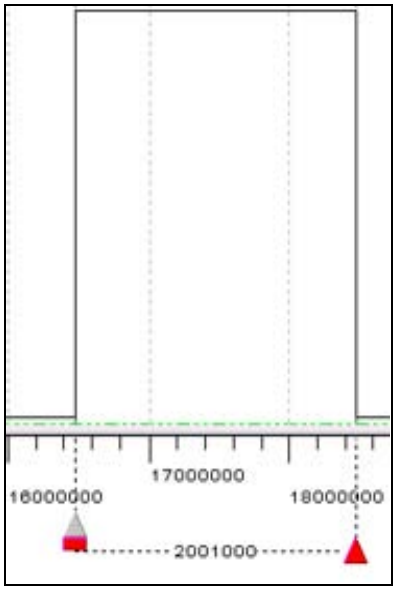
RESULTS:

Waveform analyzer and oscilloscope both verified the correctness of the servomotor control code. Used 1000 clock rate and 40000 points to capture the results.

Waveform Analyzer pulses of widths 1msec, 1.5msec, and 2msec:







Note: The additional 0.001 millisecond delay can be contributed to the lack of desired screen resolution to accurately align markers with pulse edges.

Hardware simulation with Futaba model number 3003 servomotor was also successful. The motor behaved according to specifications and responded correctly to the state of the switches on the RC-200E device.

Appendix A: Waveform Analyzer Example Source Code

```
//WaveformExample.hcc

set clock = external "P1"
// DKSync.dll is used to synchronize several simulations
// In this instance, it is used to synchronize PAL simulation with
// the waveform analyzer.
with { extlib = "DKSync.dll", extinst = "50", extfunc = "DKSyncGetSet" };
unsigned 3 x = 0;

interface bus_out() ob1(unsigned 3 out = x)

// DKConnect.dll is used to pass data between several simulations.
// In this instance it is used to pass the pin output data
// to the waveform analyzer.
with {extlib = "DKConnect.dll", extinst = "t(3)", extfunc =
"DKConnectGetSet"};

void main(void)
{
  while(1) x++;
}
```

http://zdex.net/scripts/viewcpp.asp?name=/HCC/WaveFormExample.hcc

Appendix B: Generate Pulses Source Code.

```
// GeneratePulses.hcc - Generate Pulses
/*************************************************************
* Project    :    Generate Pulses                             *
* Date       :    1 Mar 2004                                  *
* File       :    GeneratePulses.hcc                          *
* Author     :    Sergey Averchenkov                          *
*************************************************************
* Desc       :    This project endlessly generates a 1 msec   *
*            :    pulse every 20 msec on the output pin #3    *
*            :    on the expansion header.                    *
*************************************************************/

#if (defined USE_RC200 || USE_RC200E)
#define PAL_TARGET_CLOCK_RATE 50000000
#else
#define PAL_ACTUAL_CLOCK_RATE 1000000
```

```
  set clock = external "P1"
    with
    {
      extlib  = "DKSync.dll",
      extinst = "1000",       // Period of 1MHz simulated clock
      extfunc = "DKSyncGetSet"
    };
  #endif

  #include <pal_master.hch>
  #include <stdlib.hch>

  unsigned 1 pulse = 0;
  // "M2" is the pin #3 on the RC-200E device.
  interface bus_out () OutBus(unsigned 1 OutPort=pulse)
        with
        {
      #if (defined USE_RC200 || USE_RC200E)
          data  = { "M2" }
      #else
          extlib  = "DKConnect.dll",
          extinst = "t(1)",
          extfunc = "DKConnectGetSet"
      #endif
        };


  // microsec_delay()
  // ----------------------------------------------------------------
  /*
   *    Pauses the execution for msec microseconds.
   */
  macro proc microsec_delay(msec);

  // main()
  // ----------------------------------------------------------------
  /*
   *    This thread generates pulses.
   */
  void main(void)
  {
    while(1)
    {
      pulse = 0;
      microsec_delay(20000);
      pulse = 1;
      microsec_delay(1000);
    }
  }

  // microsec_delay()
  // ----------------------------------------------------------------
  /*
   *    Pauses the execution for msec microseconds.
   */
  static macro proc microsec_delay(msec)
  {
    macro expr cycles = (PAL_ACTUAL_CLOCK_RATE * msec) / 1000000;
    unsigned (log2ceil(cycles)) count;
```

```
  count = 0;
  do
  {
    count++;
  }
  while(count != cycles - 1);
}
```

http://zdex.net/scripts/viewcpp.asp?name=/HCC/GeneratePulses.hcc

Appendix C: Servomotor Source Code.

```
// Servomotor.hcc
/*****************************************************************
* Project   :    Servomotor Control                                *
* Date      :    1 Mar 2004                                        *
* File      :    Servomotor.hcc                                    *
* Author    :    Sergey Averchenkov                                *
*****************************************************************
* Desc      :    This program generates a pulse every 20 msec.  *
*           :    If one pushbutton is pressed, the pulse is 1.0 *
*           :    msec long; if the other button is pressed,     *
*           :    the pulse is 2.0 msec long. If neither button  *
*           :    (or both buttons) is pressed, the pulse is     *
*           :    1.5 msec long. This program is used to control *
*           :    a servomotor Futaba model number 3003, which   *
*           :    rotates clockwise when it receives 1.0 msec    *
*           :    pulses, counterclockwise when it receives 2.0  *
*           :    msec pulses, and not at all when it receives   *
*           :    1.5 msec pulses.                               *
*****************************************************************/

#if (defined USE_RC200 || USE_RC200E)
#define PAL_TARGET_CLOCK_RATE 50000000
#define SW0 0
#define SW1 1
#define NUM_SWITCHES 2
#else
#define PAL_ACTUAL_CLOCK_RATE 1000000
set clock = external "P1"
  with
  {
    extlib  = "DKSync.dll",
    extinst = "1000",       // Period of 1MHz simulated clock
    extfunc = "DKSyncGetSet"
  };
#define SW0 8
#define SW1 9
#define NUM_SWITCHES 10
#endif

#include <pal_master.hch>
#include <stdlib.hch>

chan <unsigned 1> send_pulse;
```

```
unsigned 1 pulse = 0;
// "M2" is the pin #3 on the RC-200E expansion header.
interface bus_out () OutBus(unsigned 1 OutPort=pulse)
      with
      {
    #if (defined USE_RC200 || USE_RC200E)
        data  = { "M2" }
    #else
        extlib  = "DKConnect.dll",
        extinst = "t(1)",
        extfunc = "DKConnectGetSet"
    #endif
      };

unsigned 1 switchZeroPressed=0;
unsigned 1 switchOnePressed=0;

// microsec_delay()
// -------------------------------------------------------------
/*
 *    Pauses the execution for msec microseconds.
 */
macro proc microsec_delay(msec);

// main()
// -------------------------------------------------------------
/*
 *    This thread reads the state of the switches.
 */
void main(void)
{
  PalVersionRequire (1, 0);
  while(1)
  {
    par
    {
      PalSwitchRead(PalSwitchCT(SW0), &switchZeroPressed);
      PalSwitchRead(PalSwitchCT(SW1), &switchOnePressed);
    }
  }
}


// main()
// -------------------------------------------------------------
/*
 *   This thread generates synchronization pulse every
 *   20 milliseconds.
 *   Handel-C channels guarantee that all threads that access
 *   a channel will do so in the same clock cycle.
 *   This allows us to synchronize two threads by first sending
 *   a bit to a channel every 20 msec and then reading it
 *   from a different thread. Both write and read operations will
 *   occur in the same clock cycle, so two threads will be
 *   appear to be synchronized.
 */
void main(void)
{
  while(1)
```

```
    {
      microsec_delay(20000);
      send_pulse ! 1;
    }
  }

  // main()
  // ---------------------------------------------------------------
  /*
   *   This thread generates pulses based on the states
   *   of the switches. Waits for synchronization pulse.
   */
  void main(void)
  {
    unsigned 1 ground;

    while(1)
    {
      pulse = 0;
      // We simply discard the value of the
      // channel. It is only used for purposes
      // of synchronization.
      send_pulse ? ground;
      pulse = 1;
      if(switchOnePressed)
      {
        // 1 1
        if(switchZeroPressed)
        {
          microsec_delay(1500);
        }
        // 1 0
        else
        {
          microsec_delay(2000);
        }
      }
      else
      {
        // 0 1
        if(switchZeroPressed)
        {
          microsec_delay(1000);
        }
        // 0 0
        else
        {
          microsec_delay(1500);
        }
      }
    }
  }

  // microsec_delay()
  // ---------------------------------------------------------------
  /*
   *    Pauses the execution for msec microseconds.
   */
  static macro proc microsec_delay(msec)
```

```
{
  macro expr cycles = (PAL_ACTUAL_CLOCK_RATE * msec) / 1000000;
  unsigned (log2ceil(cycles)) count;

  count = 0;
  do
  {
    count++;
  }
  while(count != cycles - 1);
}
```

http://zdex.net/scripts/viewcpp.asp?name=/HCC/Servomotor.hcc