X-Bus Intensivseminar

ExDac Echtzeitprogramm

SORCUS Computer GmbH
Dipl.-Ing. Jens Daneke

SORCUS

Dieses Dokument beschreibt den Aufbau des Echtzeitprogramms, das die Messung und Pufferung der Messwerte übernimmt.

Vorgehensweise

- Programm-Gerüst mit RTDS erzeugen
- main-Prozedur ausfüllen (PDT Initialisierung)
- Globale Parameter definieren
- Parameter initialisieren in der auto_init
- Konfigurationsfunktion als Schnittstelle zum PC-Programm programmieren (Handles für Host, MDDs und Puffer holen, Kanäle öffnen)
- Start und Stop Prozedur
- Mess-Prozedur

Programm anpassen

- PDT-Initialisierung anpassen:
  - Programm Nummer = 9008 hex
  - Version =
  - Task-Typ und Interrupt = Interrupt-Task unter Timer A
  - Datenbereich = nicht benötigt
  - Parameterbereich = im Programm enthalten
  - Prozeduren und Funktionen hinzufügen:
    - Start (Prozedur #3)
    - Stop (Prozedur #4)
    - Konfiguration (Funktion #2)

Parameter

- im Parameterbereich werden folgende Variablen gespeichert:
  - Konfigurations-Parameter wie Abtastrate, Messwert-Anzahl
  - Handles und Variable zur Verwaltung des Messablaufs
  - Verwendete SRQ-Worte
- Die Verwendung des Parameterbereiches für diese globalen Variablen (d.h. allen Prozeduren des Programms zur Verfügung stehenden) erleichtert die Fehlersuche und ermöglicht einfache Parametrierung des Programms durch Veränderung der Parameter von außen
- Der Parameterbereich wird aus Geschwindigkeitsgründen als globale Variable des Programms angelegt. Dadurch kann das Programm direkt darauf zugreifen.

Echtzeit-Konfiguration

- Folgende Aufgaben muss das Echtzeitprogramm für seine Konfiguration erledigen:
  - Handle der eigenen CPU holen
  - Handle der Host-CPU holen (zum SRQ versenden)
  - Puffer-Handle holen
  - MDD-Handle holen
  - Kanäle öffnen
- Konfiguriert wird in der Funktion 2:

```
void MAXEXPORT MeasureInit
      (USHORT *pusInsize,            // = sizeof (DAQ_INIT_DATA)
       DAQ_INIT_DATA *prcInit,       // Konfigurationsdaten
       USHORT *pusOutsize,           // = 0 (keine Rückgabedaten)
       USHORT *pusDataOut);          // keine Rückgabedaten
```

Konfiguration - Schnittstelle

- Die Konfigurationsfunktion-Funktion bekommt die notwendigen Daten vom Windows-Programm in folgender Struktur übergeben:

```
typedef struct
{
   USHORT      usADSlot;          // slot number of the X-AD14-20
   USHORT      usTimeValue;       // for aquisition frequency: 1 = 0,84us
   ULONG       ulBufferID;        // ID for used OsX buffer
   ULONG       ulAcquisitions;    // number of data acquisitions cycles
} DAQ_INIT_DATA;
```

Konfigurations-Funktion 1

```
void MAXEXPORT MeasureInit(USHORT *pusInsize, DAQ_INIT_DATA *prcInit,
                           USHORT *pusOutsize, USHORT *pusDataOut)
{
   CPS_XAD1420 CPS_Ain;
   ULONG       ulValid;

   max_entry_func();

   // Save measurement parameter
   parameter.usTimerRate  = prcInit->usTimeValue;
   parameter.ulMaxSamples = prcInit->ulAcquisitions;

   // get a handle for the host PC (to send SRQs)
   parameter.Error = max_connect_cpu (parameter.usHostNo, 0, 0, NULL,
                                      &parameter.hHost);

   // get a handle for the buffer
   parameter.Error = max_get_buffer_handle (parameter.hMyself,
                                            prcInit->ulBufferID,
                                            &parameter.hBuffer);
   // get total buffersize
   parameter.Error = max_get_buffer_status(parameter.hBuffer, &ulValid,
                                           &parameter.ulBufferSize);
```

Konfigurations-Funktion 2

```
    // get a handle for the X-AD14-20 MDD
    parameter.Error = max_load_mdd (parameter.hMyself,
                                    prcInit->usADSlot, 0, 0,
                                    MDD_X_AD14_20, NULL,
                                    &parameter.hADMdd);

    // open a channel for analog inputs AIN-0 and AIN-1
    CPS_Ain.usDevice     = DEVICE_AIN_SE;
    CPS_Ain.usIndexFirst = 0;
    CPS_Ain.usIndexLast  = 1;
    CPS_Ain.usRange      = RANGE_BIP_10V;
    CPS_Ain.usFlags      = _CP_UNCORRECTED;
    CPS_Ain.usReadMode   = IO_MODE_DIRECT;
    CPS_Ain.usSettleTime = 200;     // Multiple of 1 ns
    parameter.Error = max_open_channel (parameter.hADMdd,
                                        sizeof (CPS_XAD1420),
                                        &CPS_Ain, NULL, 0,
                                        &parameter.hAin);
    max_exit_func (parameter.Error);
}
```

Start Prozedur

```
void MAXEXPORT Start(void)
{
   max_entry_proc();

   // empty buffer
   max_clear_buffer(parameter.hBuffer);

   // start Timer
   parameter.Error = max_set_timer(parameter.hMyself,
                                   MAX1_TIMER_A, MAX1_TIMER_MODE_INT,
                                   parameter.usTimerRate);
   // Activate the task
   // from now on procedure #0 will be called any time Timer A expires
   parameter.Error = max_wakeup_ii_task(parameter.hMyself,
                                        parameter.rcTDT.usTaskNo);
   max_exit_proc();
}
```

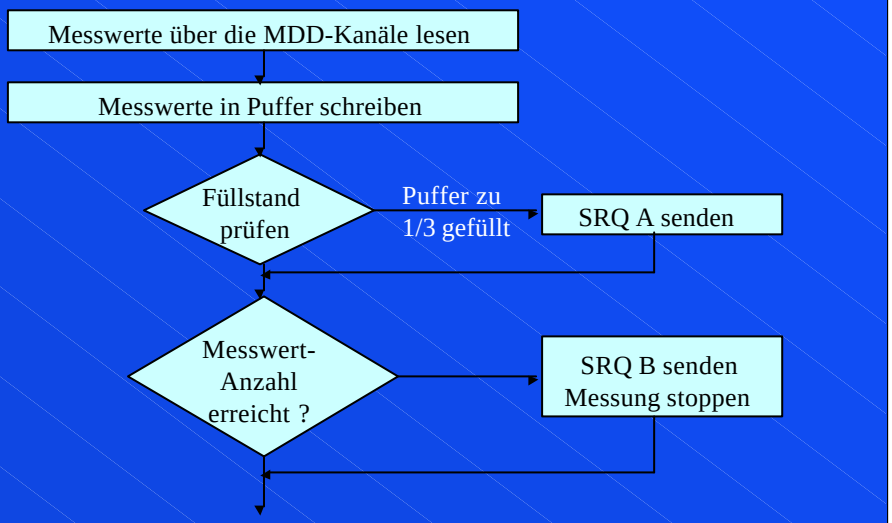
Stop Prozedur

```
void MAXEXPORT Stop(void)
{
   max_entry_proc();
   max_sleep_task (parameter.hMyself, parameter.rcTDT.usTaskNo);
   max_exit_proc();
}
```

Mess-Prozedur

- Wird mit jedem Timer A Interrupt aufgerufen
- Misst jeweils die Analogkanäle und schreibt die Werte hintereinander in den OsX-Ringpuffer (Datentyp *OSX_BUFFER_DATA*)
- Schickt einen SRQ zum Host, wenn der Puffer zu mehr als 1/3 gefüllt ist. In diesem Fall wird das SRQ-Senden solange unterbunden, bis der Puffer wieder geleert worden ist.
- Die Anzahl der Messungen wird überwacht und beim Erreichen des Endes ein SRQ versendet und die Messung gestoppt.
- Treten Fehler auf, wird ein SRQ gesendet.

Echtzeit-Messung



Mess-Prozedur 1: Werte erfassen und in Puffer schreiben

```
void MAXEXPORT MainProc(void)
{
    OSX_BUFFER_DATA MeasureData;
    ULONG  ulSize, ulFree, ulValid;

    max_entry_proc();

    // read channels
    ulSize = 8;    // 2 long values for analog inputs
    parameter.Error = max_read_channel_block(parameter.hAin,
                                             &ulSize,
                                             MeasureData.alAin);
    // write values into buffer
    ulSize = sizeof(MeasureData);
    parameter.Error = max_set_buffer (parameter.hBuffer,
                                      MAX_BUFFER_MOVE_ABS,
                                      &ulSize,
                                      &MeasureData);
```

Dieser Teil der Hauptprozedur zeigt das Lesen der Analogwerte und das Speichern im OsX Puffer.

In diesem Auszug wird das Senden von SRQs im Fehlerfall aus Gründen der Übersichtlichkeit nicht gezeigt.

Mess-Prozedur 2: Puffer-Füllstand überwachen

```
parameter.Error = max_get_buffer_status(parameter.hBuffer,
                                        &ulValid, &ulFree);
// check if SRQ has to be sent
if (parameter.usReleaseSRQ)
{
   // no SRQ has been sent since buffer is filled more than 1/3
   if (ulValid >= parameter.ulBufferSize / 3)
   {
      parameter.usReleaseSRQ = 0;
      // if buffer is filled more than 1/3 send SRQ to the host
      parameter.Error = max_send_srq(parameter.hHost, 0,
                                     parameter.usBufferFullSRQ);
   }
}
else
{  // SRQ has been sent already since buffer is filled more than 1/3
   if (ulValid < parameter.ulBufferSize / 3)
      // enable SRQs again if buffer state falls below 1/3
      parameter.usReleaseSRQ = 1;
}
```


Dieser Teil der Hauptprozedur zeigt die Überwachung des Puffers.

Ist er zu mehr als 1/3 gefüllt, wird ein SRQ gesendet. Mit dem Flag usReleaseSRQ wird das Senden von SRQs solange unterbunden, bis der Puffer wieder zu weniger als 1/3 gefüllt ist (also nachdem er vom Host-PC geleert wurde).

Dieser Teil könnte zu Optimierungszwecken in eine separate NI-Task ausgelagert werden, da es nicht in jedem Messzyklus erforderlich ist den Puffer zu prüfen. Diese Überprüfung reicht gelegentlich, wenn der Puffer gross genug ist.

Mess-Prozedur 3
Anzahl der Messungen überwachen

```
   // increment measurement counter
   parameter.ulSampleCount++;

   // check if desired number of data acquisitions is done
   if (parameter.ulSampleCount == parameter.ulMaxSamples)
   {
      // if number of acquisitions is reached
      parameter.Error = max_send_srq(parameter.hHost, 0,
                                     parameter.usFinishSRQ);
      // call stop procedure to stop measurement
      max_call_proc(parameter.hMyself, parameter.rcTDT.usTaskNo, 4);
   }
   max_exit_proc();
}
```

SORCUS

Dieser Teil der Hauptprozedur zeigt die Überwachung der Anzahl der Messzyklen. Bei der eingestellten Anzahl wird ein SRQ gesendet und die Messung gestoppt.

Performance

- Dauer der Messroutine inkl. Taskswitch: ~50us

→Messrate 20 kHz für 2 Analogkanäle im normierten Format

- Durch Optimierungen sind für diesen Fall max. 50 kHz erreichbar

Optimierungsmöglichkeiten 1

- Analogwerterfassung beschleunigen:
  - Werte im Hardware Format erfassen
- Bibliotheksfunktionen zur Steigerung der CPU-Performance:
  - `max_set_error_check_level(0);`
  -
```
usMode = 3;
max_cache_control(parameter.hMyself, &usMode);
```
  -
```
max_cpu_speed_control (parameter.hMyself,
                       MAX1_CPU_HYPER_SPEED,
                       MAX1_SPEED_100MHZ);
```

Optimierungsmöglichkeiten 2

- so wenig Bibliotheks Funktionen wie möglich aufrufen
- Puffer-Füllstand in einer parallel laufenden NI-Task prüfen und von dort SRQs verschicken.
- im Extremfall anstelle der Puffer-Funktionen einen Speicherbereich für die Messdaten reservieren (max_allocate_ram) und die Daten mit inline-Assembler Befehlen dort eintragen:

```
_AX = 0;
_FS = _AX;              // fs = 0 setzen
_EAX = ulData;          // zu schreibende Daten
_EBX = ulMemoryAddress;// Speicheradresse, in die geschrieben wird
asm mov fs:[ebx], eax; // Daten schreiben
```

SORCUS

Beim direkten Schreiben in den Speicher ist zu beachten, dass Zeiger in C oder Pascal nur bis 1 Megabyte verwendet werden dürfen.

Da das CPU-Modul über mehr Speicher verfügt, müssen für Zugriffe entweder die Bibliotheksfunktionen max_read_memory und max_write_memory verwendet werden oder es muss mit Hilfe der oben gezeigten Assembler-Sequenz gearbeitet werden. Damit können Daten in jede physikalische Adresse im gesamten Speicherraum des X-MAX-1 geschrieben werden.

Wo der Speicher reserviert wird, kann beim Allokieren mit max_allocate_ram angegeben werden.

Optimierungsmöglichkeiten 3

- Für jeden Kanal ein eigenes CPU-Modul und ein eigenes X-AD14-20 Modul verwenden, das mit nichts anderem beschäftigt ist als einen einzigen Analogkanal zu erfassen. Dafür gibt es im MDD des X-AD14-20 einen High-Speed Mode, in dem die maximal mögliche Abtastrate erreicht werden kann.

Ende