CHALMERS

EDA222 Real time systems

Case: Programming the HCS12 CAN controller

Contents:

Introduction

In this document we will demonstrate low level programming of a non-trivial interface. We use the CAN controller present in the Freescale HCS12 family of microcontrollers.

The demonstration consists of three parts; we first show an ANSI-C implementation. For ease of comprehension the implementation itself is the simplest possible, but still working solution. It is not meant to be optimal and it is, in fact, of less use in a real-time system since “busy wait”-technics are used.

We then continue and demonstrate how these routines could be used from an Ada95 (GNAT) application. Note that we have not been able to test this in practice due to lack of a working Ada95/HCS12 real-time environment.

We finally rewrite the software package entirely in Ada95, and again, note that we have not been able to test the solution in practice.

Basic software: ANSI-C

Basic definitions and types:
The HCS12DG256 has two on-chip CAN-controllers, called “can0” and “can4”. The controllers are identical except from their different base addresses in the memory map. In ANSI-C it is therefore convenient to describe the controller registers with a struct type, write software that manipulates the registers simply through a pointer, pointing at the first register (a base pointer).

```
#define  CAN0BASE 0x0140
#define  CAN4BASE 0x0280

typedef struct   canTAG{
        unsigned char canctl_0;   // 0
        unsigned char canctl_1;   // 1
        unsigned char canbtr_0;   // 2
        unsigned char canbtr_1;   // 3
        unsigned char canrflg;    // 4
        unsigned char canrier;    // 5
        unsigned char cantflg;    // 6
        unsigned char cantier;    // 7
        unsigned char cantarq;    // 8
        unsigned char cantaak;    // 9
        unsigned char cantbsel;   // 0xA
        unsigned char canidac;    // 0xB
        unsigned char Reserved0[2]; // 0xC
        unsigned char canrxerr;   // 0xE
        unsigned char cantxerr;   // 0xF

        unsigned char    canidar_0; // 0x10
        unsigned char    canidar_1; // 0x11
        unsigned char    canidar_2; // 0x12
        unsigned char    canidar_3; // 0x13
        unsigned char    canidmr_0; // 0x14
        unsigned char    canidmr_1; // 0x15
        unsigned char    canidmr_2; // 0x16
        unsigned char    canidmr_3; // 0x17
        unsigned char    canidar_4; // 0x18
        unsigned char    canidar_5; // 0x19
        unsigned char    canidar_6; // 0x1A
        unsigned char    canidar_7; // 0x1B
        unsigned char    canidmr_4; // 0x1C
        unsigned char    canidmr_5; // 0x1D
        unsigned char    canidmr_6; // 0x1E
        unsigned char    canidmr_7; // 0x1F
        unsigned char    canrxfg; //  0x20
```

```
        unsigned char    canridr_0; // 0x20
        unsigned char    canridr_1; // 0x21
        unsigned char    canridr_2; // 0x22
        unsigned char    canridr_3; // 0x23
        unsigned char    canrdsr_0; // 0x24
        unsigned char    canrdsr_1; // 0x25
        unsigned char    canrdsr_2; // 0x26
        unsigned char    canrdsr_3; // 0x27
        unsigned char    canrdsr_4; // 0x28
        unsigned char    canrdsr_5; // 0x29
        unsigned char    canrdsr_6; // 0x2A
        unsigned char    canrdsr_7; // 0x2B
        unsigned char    canrdlr; // 0x2C
        unsigned char    canrts;  // 0x2E
        unsigned char    canrtsrh; // 0x2E
        unsigned char    canrtsrl; // 0x2F
        unsigned char    cantxfg; // 0x30

        unsigned char    cantidr_0; // 0x30
        unsigned char    cantidr_1; // 0x31
        unsigned char    cantidr_2; // 0x32
        unsigned char    cantidr_3; // 0x33
        unsigned char    cantdsr_0; // 0x34
        unsigned char    cantdsr_1; // 0x35
        unsigned char    cantdsr_2; // 0x36
        unsigned char    cantdsr_3; // 0x37
        unsigned char    cantdsr_4; // 0x38
        unsigned char    cantdsr_5; // 0x39
        unsigned char    cantdsr_6; // 0x3A
        unsigned char    cantdsr_7; // 0x3B
        unsigned char    cantdlr; // 0x3C
        unsigned char    canttbtr; // 0x3D
        unsigned char    cantbsr; // 0x3E
        unsigned char    canttsrh; // 0x3E
        unsigned char    canttsrl; // 0x3F

} CAN, *PCAN ;
```

To further increase readability it is advisable to define bitmasks for frequently used bits in the various registers:

```
/*
        CAN register bit
definitions
*/
#define RXFRM      0x80
#define RXACT      0x40
#define CSWAI      0x20
#define SYNCH      0x10
#define TIME       0x08
#define WUPE       0x04
#define SLPRQ      0x02
#define INITRQ     0x01

#define CANE       0x80
#define CLKSRC     0x40
#define LOOPB      0x20
#define LISTEN     0x10
#define WUPM       0x04
#define SLPAK      0x02
#define INITAK     0x01

#define SJW1       0x80
#define SJW0       0x40
#define BRP5       0x20
#define BRP4       0x10
#define BRP3       0x08
#define BRP2       0x04
#define BRP1       0x02
#define BRP0       0x01

#define SAMP       0x80
#define TSEG22     0x40
#define TSEG21     0x20
#define TSEG20     0x10
#define TSEG13     0x08
```

```
#define TSEG12     0x04
#define TSEG11     0x02
#define TSEG10     0x01

#define WUPIF      0x80
#define CSCIF      0x40
#define RSTAT1     0x20
#define RSTAT0     0x10
#define TSTAT1     0x08
#define TSTAT0     0x04
#define OVRIF      0x02
#define RXF        0x01

#define WUPIE      0x80
#define CSCIE      0x40
#define RSTATE1    0x20
#define RSTATE0    0x10
#define TSTATE1    0x08
#define TSTATE0    0x04
#define OVRIE      0x02
#define RXFIE      0x01

#define TXE2       0x04
#define TXE1       0x02
#define TXE0       0x01

#define TXEIE2     0x04
#define TXEIE1     0x03
#define TXEIE0     0x01

#define ABTRQ2     0x04
#define ABTRQ1     0x02
#define ABTRQ0     0x01

#define ABTAK2     0x04
```

```
#define ABTAK1     0x02
#define ABTAK0     0x01

#define TX2        0x04
#define TX1        0x02
#define TX0        0x01

#define IDAM1      0x20
#define IDAM0      0x10
#define IDHIT2     0x04
#define IDHIT1     0x02
#define IDHIT0     0x01

#define RXERR7     0x80
#define RXERR6     0x40
#define RXERR5     0x20
#define RXERR4     0x10
#define RXERR3     0x08
#define RXERR2     0x04
#define RXERR1     0x02
#define RXERR0     0x01

#define TXERR7     0x80
#define TXERR6     0x40
#define TXERR5     0x20
#define TXERR4     0x10
#define TXERR3     0x08
#define TXERR2     0x04
#define TXERR1     0x02
#define TXERR0     0x01
```

We now need routines to initialize, send and receive. The following code shows an example of how to setup the controllers and then provide functions for transmitting, setting up a controller for receiving and finally actually receive data from the CAN-bus.

```
void can_init( PCAN port )
        /* init CAN controller, port is the controller base */
{
        port->canctl_1 = 0xC0;                  /* enable, clock source bus clock */

        /* INIT MODE */
        while(( port->canctl_0 & RXACT));
        port->canctl_0 = SLPRQ;                 /* sleep mode request */
        while(!(port->canctl_1 & SLPAK));       /* wait for sleep mode acknowledge */
        port->canctl_0 = SLPRQ | INITRQ;        /* init mode request */
        while(!(port->canctl_1 & INITAK));      /* init mode acknowledge */

        port->canbtr_0 = 0;                     /* Set CAN bus bit rate to 1 Mbit/sec */
        port->canbtr_1 = 0x14;

        port->canidar_0 = 0xFF;                 /* acceptance filtering, */
        port->canidar_1 = 0xFF;                 /* for now accept ANYTHING */

        port->canidar_2 = 0xFF;
        port->canidar_3 = 0xFF;
        port->canidar_4 = 0xFF;
        port->canidar_5 = 0xFF;
        port->canidar_6 = 0xFF;
        port->canidar_7 = 0xFF;

        port->canctl_0 = 0x00;                  /* normal mode request */
        while( port->canctl_1 & INITAK);        /* normal mode acknowledge */
        while(( port->canctl_1 & SLPAK));       /* out of sleep mode */
        /* END OF INIT MODE */
}
```

```
void    can_setup_receive(char *filter,  CAN *c  )
        /* parameters:
                filter[4] contains message filter
                port is the CAN port  */
{
        unsigned char * pctr;
        /* INIT MODE */
        while(( c->canctl_0 & RXACT));
        c->canctl_0 = SLPRQ;                    /* sleep mode request */
        while(!(c->canctl_1 & SLPAK));          /* sleep mode acknowledge */
        c->canctl_0 = SLPRQ | INITRQ;           /* init mode request */
        while(!(c->canctl_1 & INITAK));         /* init mode acknowledge */

        c->canidac = 0x00;                      /* 32 bit acceptance filter */

        pctr = &(c->canidar_0);                 /* acceptance filter bank 0 */
        *pctr++ = 0x00;
        *pctr++ = 0x00 | 0x18;
        *pctr++ = 0x00;
        *pctr =   0x00;

        pctr = &(c->canidmr_0);                 /* message filter bank0, must match */
        *pctr++ = filter[0];
        *pctr++ = filter[1];
        *pctr++ = filter[2];
        *pctr++ = filter[3];


        *pctr++ = filter[4];                    /* message filter bank 1, don't care */
        *pctr++ = filter[5];
        *pctr++ = filter[6];
        *pctr =   filter[7];

        c->canctl_0 = 0x00;                     /* normal mode request */
        while(( c->canctl_1 & INITAK));         /* normal mode acknowledge */
        while(( c->canctl_1 & SLPAK));          /* out of sleep mode */
        /* END OF INIT MODE */
}
```

```
void    can_receive(  char *buffer, int *len, PCAN port )
        /* parameters:
           port is the CAN port
           buffer must be large enough to hold a message on successfull receive
           len is message bytecount (0..8) */
{
        unsigned char * pctr;
        unsigned char  count;

        if(!(port->canrflg & RXF))    /* message received? */
        {
                *len = 0;
                return;               /* No message */
        }

        pctr = &(port->canrdlr);      /* Get message byte count */
        count = *pctr;
        *len = (int) count;

        pctr = &(port->canrdsr_0);    /* Collect the message ... */
        while( count )
        {
                *buffer++ = *pctr++;
                count--;
        }
        port->canrflg = RXF;          /* Ack. by clearing flag */

}
```

```
void    can_send( char* id, char *buffer, int len, PCAN port)

        /* parameters:
           id[4] contains identifier
           buffer[len] contains the message
           len is message bytecount
           port is the CAN port */

{
        unsigned char * pctr;
        unsigned char  count;
        unsigned char  txbuf;

        /* get free Tx buffer */
        do{
                txbuf = port->cantflg;
        } while (txbuf == 0 );
        port->cantbsel = txbuf;
        txbuf = port->cantbsel;       /* select buffer */

        port->cantidr_0 = id[0];      /* Setup the identifier registers... */
        port->cantidr_1 = id[1] | 0x18;
        port->cantidr_2 = id[2];
        port->cantidr_3 = id[3] & 0xFE;

        count = (unsigned char) len;  /* Setup the data registers... */
        pctr = &(port->cantdsr_0);
        port->cantdlr = count;
        while( count )
        {
                *pctr++ = *buffer++;
                count--;
        }

        port->cantflg = txbuf;        /* clear flag to start transmission */

        while(( port->cantflg & txbuf ));     /* wait for transmission to complete */
}
```

Interfacing Ada95 to ANSI-C routines

Interfacing to routines in an existing C-library is straightforward. Supply appropriate type definitions and do the “imports”.

```
-- can_driver.ads
with    interfaces.c;

package CAN_DRIVER is
        type Canfilter_Type is array (1..4) of interfaces.c.unsigned_char;
        type Canfilter_Ptr_Type is access all Canfilter_Type;

        type Canid_Type is array (1..4) of interfaces.c.unsigned_char;
        type Canid_Ptr_Type is access all Canid_Type;

        type Canbuffer_Type is array (1..8) of character;
        type Canbuffer_Ptr_Type is access all Canbuffer_Type;

        type Int_Ptr_Type is access all integer;

        type Can_Ptr_Type is access all character;
        pragma Convention( C , Can_Ptr_Type );

        procedure A_can_init( Can_Ptr : Can_Ptr_Type );
        pragma Import( C , A_can_init , "can_init");

        procedure A_can_setup_receive( filter : Canfilter_Ptr_Type ;
                        Can_Ptr : Can_Ptr_Type );
        pragma Import( C , A_can_setup_receive , "can_setup_receive" );

        procedure A_can_receive( buffer : out Canbuffer_Ptr_Type ;
                        len : in out Int_Ptr_Type ; Can_Ptr : Can_Ptr_Type );
        pragma Import( C , A_can_receive , "can_receive" );

        procedure A_can_send( id : Canid_Ptr_Type ; buffer : Canbuffer_Ptr_Type ;
                        len : in integer ; Can_Ptr : Can_Ptr_Type );
        pragma Import( C , A_can_send , "can_send" );

end CAN_DRIVER;
```

The following test program illustrates how the package is used:

```
--
-- can_driver_test.adb
--

with CAN_DRIVER; use CAN_DRIVER;
with System.Storage_Elements;


procedure can_driver_test is
        cptr0   : aliased Can_Ptr_Type;
        for cptr0'Address use System.Storage_Elements.To_Address(16#0140#);
        cptr1   : aliased Can_Ptr_Type;
        for cptr1'Address use System.Storage_Elements.To_Address(16#0280#);
        buffer  : aliased Canbuffer_Type;
        rbuffer : Canbuffer_Ptr_Type;
        filter  : aliased Canfilter_Type;
        canid   : aliased Canid_Type;
        len     : aliased integer;
        plen    : Int_Ptr_Type;
begin
        --some initial values...
        filter  := (16#FF#,16#FF#,16#FF#,16#FF#);
        canid   := (16#FF#,16#FF#,16#03#,16#64#);

        -- setup controllers
        A_can_init ( cptr0 );  -- Init controller 0
        A_can_init ( cptr1 );  -- Init controller 1

        -- setup controller 1 as receiver
        A_can_setup_receive( filter'unchecked_access , cptr1);

        -- send CAN frame with controller 0
        buffer  := ('1','2','3','4','5','6','7','8');
        A_can_send( canid'unchecked_access , buffer'unchecked_access , 8 , cptr0 );

        -- wait until we get it ...
        len := 0;
        rbuffer := buffer'unchecked_access;
        plen := len'unchecked_access;
        loop
                A_can_receive( rbuffer , plen , cptr1 );
                -- 'len' is assigned through 'plen' ...
                exit when len /= 0 ;
        end loop;

end;
```

Rewrite: CAN software package in Ada95

The major obstacle here is writing all 32 register (and bit) definitions. After this has been accomplished, the task is almost trivial. We therefore show only the first register definitions, and the example how to convert the original C-program to its Ada counterpart.

```
-- can_driver_rewrite.ads
package  CAN_DRIVER_REWRITE is
         --types that's only used within the driver package
         type BIT is (CLEAR, SET );
         for BIT use (CLEAR => 0, SET => 1);
         for BIT'Size use 1;
         -- type/bit definitions for can controller
         type Atype_canctl_0 is            -- Control register 0
         record
                  RXFRM      : BIT;
                  RXACT      : BIT;
                  CSWAI      : BIT;
                  SYNCH      : BIT;
                  TIME       : BIT;
                  WUPE       : BIT;
                  SLPRQ      : BIT;
                  INITRQ     : BIT;
         end record;
         for Atype_canctl_0 use
         record
                  RXFRM      at 0 range 0..0;
                  RXACT      at 0 range 1..1;
                  CSWAI      at 0 range 2..2;
                  SYNCH      at 0 range 3..3;
                  TIME       at 0 range 4..4;
                  WUPE       at 0 range 5..5;
                  SLPRQ      at 0 range 6..6;
                  INITRQ     at 0 range 7..7;
         end record;
         for Atype_canctl_0'size use 8;
         type Atype_canctl_1 is            -- Control register 1
         record
                  CANE      : BIT;
                  CLKSRC    : BIT;
                  LOOPB     : BIT;
                  LISTEN    : BIT;
                  WUPM      : BIT;
                  SLPAK     : BIT;
                  INITAK    : BIT;
         end record;
         for Atype_canctl_1'size use 8;
         record
                  CANE      at 0 range 0..0;
                  CLKSRC      at 0 range 1..1;
                  LOOPB      at 0 range 2..2;
                  LISTEN      at 0 range 3..3;
                  WUPM       at 0 range 5..5;
                  SLPAK      at 0 range 6..6;
                  INITAK     at 0 range 7..7;
         end record;
         for Atype_canctl_1'size use 8;
                  -- TODO: the remaining 30 registers...

         type CAN_Type is
         record
                  Canctl_0 : Atype_canctl_0;
                  Canctl_1 : Atype_canctl_1;
                  -- TODO: add remaining 30 registers ...
         end record;


         type UINT8 is new integer range 0..255;
         for UINT8'size use 8;
         -- types that should be visible for calling routines
         type Canfilter_Type is array (1..4) of UINT8;
         type Canfilter_Ptr_Type is access all Canfilter_Type;
         type Canid_Type is array (1..4) of UINT8;
         type Canid_Ptr_Type is access all Canid_Type;
         type Canbuffer_Type is array (1..8) of character;
         type Canbuffer_Ptr_Type is access all Canbuffer_Type;
         type Int_Ptr_Type is access all integer;
         type Can_Ptr_Type is access all CAN_Type;

         -- driver routines exported from this package are:
         procedure A_can_init( Can_Ptr : Can_Ptr_Type );
         procedure A_can_setup_receive( filter : Canfilter_Ptr_Type ;
                           Can_Ptr : Can_Ptr_Type );
         procedure A_can_receive( buffer : out Canbuffer_Ptr_Type ; len : in out
                           Int_Ptr_Type ; Can_Ptr : Can_Ptr_Type );
         procedure A_can_send( id : Canid_Ptr_Type ; buffer : Canbuffer_Ptr_Type ;
                           len : in integer ; Can_Ptr : Can_Ptr_Type );

end CAN_DRIVER_REWRITE;
```

The “body” is only sketched here; we show the basics of “can_init” and leave the rest for the reader.

```
--
-- can_driver_rewrite.adb
--
with unchecked_conversion;

package  body CAN_DRIVER_REWRITE is
         subtype int8 is integer range 1..255;
         function from_byte is new unchecked_conversion( int8, Atype_canctl_1 );
         function from_byte is new unchecked_conversion( int8, Atype_canctl_0 );

         procedure A_can_init( Can_Ptr : Can_Ptr_Type )is
         begin
                  Can_Ptr.Canctl_1 := from_byte( 16#C0# ); -- enable clock source bus
                  loop exit when Can_Ptr.Canctl_0.RXACT = SET; end loop; -- wait for controller
                  Can_Ptr.Canctl_0.SLPRQ := SET;    --sleep mode request
                  loop exit when Can_Ptr.Canctl_1.SLPAK = CLEAR; end loop; -- wait for sleep mode
                  Can_Ptr.Canctl_0.SLPRQ := SET; -- init mode request
                  Can_Ptr.Canctl_0.INITRQ := SET;
                  loop exit when Can_Ptr.Canctl_1.INITAK = CLEAR; end loop; -- init mode acknowledge
                  -- more init that requires further register definitions...

                  Can_Ptr.Canctl_0 := from_byte( 16#C0# ); -- normal mode request
                  loop exit when Can_Ptr.Canctl_1.INITAK = SET; end loop; -- normal mode acknowledge
                  loop exit when Can_Ptr.Canctl_1.SLPAK = SET; end loop; -- out of sleep mode
         end;

         procedure A_can_setup_receive( filter : Canfilter_Ptr_Type ; Can_Ptr : Can_Ptr_Type ) is
         begin
                  null; --TODO
         end;

         procedure A_can_receive( buffer : out Canbuffer_Ptr_Type ;
                           len : in out Int_Ptr_Type ; Can_Ptr : Can_Ptr_Type ) is
         begin
                  null; --TODO
         end;

         procedure A_can_send( id : Canid_Ptr_Type ; buffer : Canbuffer_Ptr_Type ;
                           len : in integer ; Can_Ptr : Can_Ptr_Type ) is
         begin
                  null; --TODO
         end;


end CAN_DRIVER_REWRITE;
```