Free Development Environment for Bus Coupling Units (BCUs)

for the European Installation Bus (EIB)

Martin Kögler
mkoegler@auto.tuwien.ac.at

June 13, 2005

Course of the talk

EIB - European Installation Bus

- EIB is a home and building automation system.
- Maintained by the KONNEX Association.
- Part of the KNX specification.
- Used for low speed applications like light switching and HVAC.
- Different transmission medias available (power-line, radio frequency, twisted pair).
- Data transfer with group communication / group objects
- Installations configured with common Windows integration tool called ETS.

EIB - European Installation Bus

Bus Coupling Units (BCUs)

- Standardized, generic platforms for embedded EIB devices
- Consists of a microcontroller (Freescale M68HC05), bus interface and system software in the ROM.
- Application programs loaded in the EEPROM (local or via the bus).
- Two BCU families: the older BCU 1 and the new BCU 2 family.

EIB bus interfaces (for PCs)

- BCU 1 and BCU 2 include a serial interface (PEI 16 and FT1.2 protocol)
- TPUART IC implements Layer 1 and parts of Layer 2
- EIBnet/IP tunnels EIB frames over TCP/IP

Current situation for non commercial developers

- A free SDK for the BCU 1 is available (including an assembler).
- Only commercial C compiler for M68HC05 exists.
- End-user versions of the ETS only accept certified programs.
- Some tools to access the EIB bus exists, but limited to certain bus interfaces and functions.

$\Rightarrow$ No free integrated solution for BCU 2 exists.

Goal of the project

creation of a free SDK for the BCU 2

- based on the GNU utilities (GCC, Binutils)
- provides RAD like concept (instead of a plain assembler interface)
- C is used as programming language
- includes an interface for integration tools (this will be parts of future projects)
  - no ETS interface
  - provides support for compilation at download time
- provides over different bus access devices access to same management functions
- no GUI interface

EIB bus access

- A network capable, multi user daemon (named *eibd*) was developed.
- Provides access to Layer 4 as well as complex management functions over a simple protocol.
- best effort, cooperative *vBusmonitor* mode
- The bus access is hidden by the backends:

  FT1.2
  : protocol of the serial interface of the BCU 2.

  EIBnet/IP
  : EIBnet/IP Routing and EIBnet/IP Tunneling client.

  TPUART
  : protocol of the TPUART IC. It uses the plain serial driver or a Linux kernel driver.

  PEI16
  : protocol of the serial interface of a BCU 1 using a kernel driver, which does the time critical data exchange. Experimental version using the serial driver exists.

RAD like development approach

- BCU program consists of a specification of EIB objects
  - used objects
  - their properties
  - their event handlers
- Assignment of low level objects to meta description of the function.
- The C fragments for event handlers are put in separate files.

Example (1/4)

Common definition

```
Device {
  PEIType 0;
  BCU bcu12; // use bcu20 for a BCU 2.0
  Title "Conditional negation";
```

Meta description (and assignment of low level objects)

```
  FunctionalBlock {
    Title "Conditional negation";
    ProfileID 10000;
    Interface {
      Reference { send };
      Abbreviation send;
      DPTType DPT_Bool; // same as 1.002
```

Example (2/4)

A receiving group object

```
GroupObject {
  Name recv;
  Type UINT1;
  on_update send_update;
  Title "Input";
  StateBased true;
};
```

A sending group object

```
GroupObject {
  Name send;
  Type UINT1;
  Sending true;
```

Example (3/4)

Code fragment for the event handler

```
void send_update()
{
   if(cond){
   send=recv+1;
   send_transmit();
   }
}
```

Example (4/4)

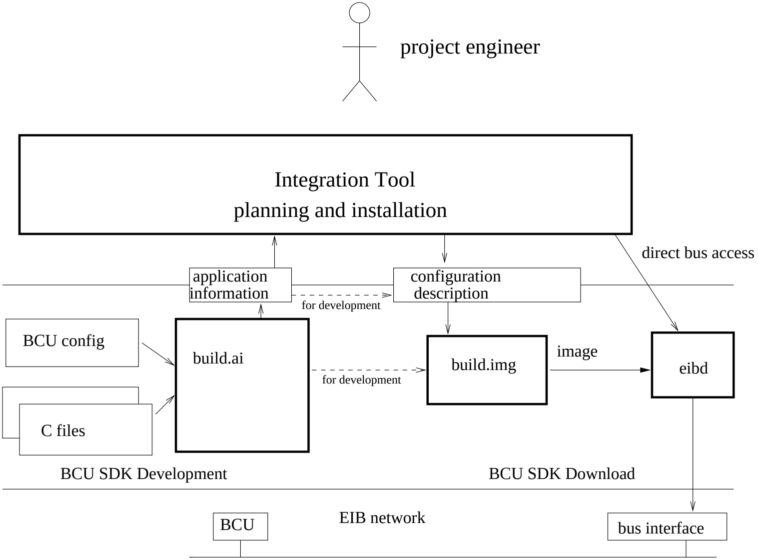
Configuration description

```
<?xml version="1.0"?>
<DeviceConfig >
  <ProgramID>xxxxxxxxxxxx</ProgramID>
  <PhysicalAddress >1.3.1</PhysicalAddress>
  <GroupObject id="id0">
    <Priority >low</Priority >
    <SendAddress >0/0/1</SendAddress>
  </GroupObject>
```

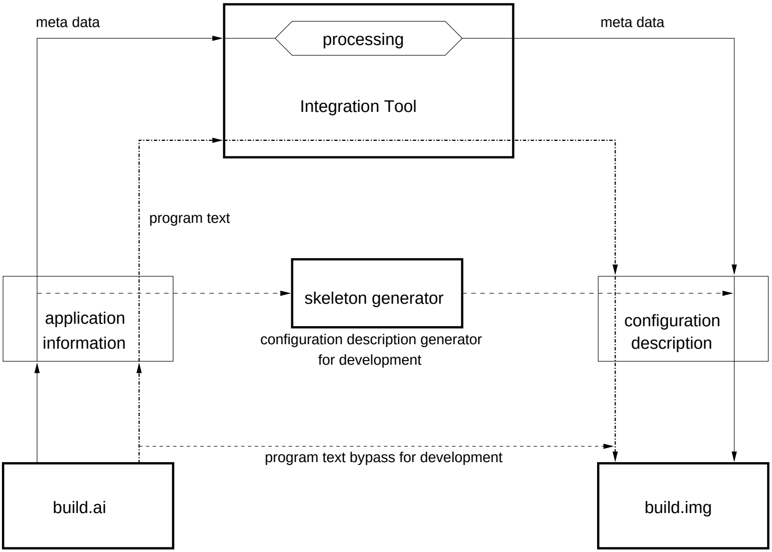
selects configuration parameters of the program

- the used individual address
- assignment of group addresses to group objects
- ...

Work flow with integration tools



Data flow


meta data
processing
Integration Tool
meta data
program text
application information
skeleton generator
configuration description generator for development
configuration description
program text bypass for development
build.ai
build.img

Port of the GNU toolchain

- the GNU toolchain was ported to the M68HC05 architecture.
- the limitations of the BCUs were used as design driver ($<$1k EEPROM, $<$ 100 Bytes RAM).
- Ported programs include:
  - Binutils (assembler, linker and object file tools)
  - GCC (GNU C compiler)
  - CPU core simulator
  - GDB frontend for the simulator
  - C runtime libraries for the simulator

- Simulator and C runtime libraries needed for GCC regression tests
- GDB for analyzing GCC generated code.
- $\Rightarrow$ incomplete

Features of the binutils port

Relaxation

- Instruction formats with different length exist.
- The needed format is often unknown at assembler runtime.
- The linker replaces longer variants if possible.
- Expanded conditional jumps are converted back, if possible.

Section movement

- The BCU 2 has non contiguous RAM sections.
- GCC needs automated distribution of variables.
- GCC prefixes each variable with a special command.
- The assembler creates a unique section.
- The linker can be instructed to move sections from a full memory region into another memory region.

GCC overview

- GNU Compiler Collection is a portable suite of compilers (about 56 supported architectures)
- language frontends for C, C++, Ada, Java, ...
- language independent middle and back end
- internal representations
  - GENERIC
  - GIMPLE
  - RTL
- uses pattern matching
- global optimizations
- per function optimizations

GNU C compiler

The M68HC05 family has several limitations:

- Two hardware registers (accumulator and index register)
- Only a small call stack
- Only *8 bit index plus fixed address* addressing mode (beside a fixed 8 or 16 bit address) but 16 bit address space.

GCC expects:

- Many GPR (general purpose registers)
- A data stack
- pointers, which can cover the entire address space

$\Rightarrow$ Emulation of missing features

small memory (BCU1: 256 byte EEPROM, 18 byte RAM) limits useable functions.

GCC internals

- 13 Bytes of RAM (RegB–RegN, reserved by BCU OS) are used as GPR.
- A byte of RAM is used as data stack pointer. Data stack starts at a 256 byte boundary. Using a different initialization value, a smaller stack area can be used.
- 16 bit pointers are emulated with self modifying code.
- mul, div and floating point operations are handled by library functions.
- Support for 1 to 8 byte integer types
- Support for transparent eeprom access (named address spaces).
- Expensive operations like setjmp/longjmp are left out.

GCC compilation process

- GCC parses a function
- GCC performs target independent optimizations on a tree representation.
- GCC converts it to high level RTL (Register Transfer Language)
  - uses only GPRs and memory locations as operands.
  - uses pseudo instructions for the 8/16/24/32/.. bit operands
- some optimizations are done
- register allocator replaces pseudo registers with GPRs and stack locations.
- splitted in low level RTL
  - each instruction corresponds to an assembler instruction or library call.
  - stack pointer cached in X register
- some optimizations are redone.
- assembler code generated

State of the GCC port

- GCC is working
  - 1335 of 36394 failed regression test cases
  - large parts fail because of insufficient memory and stack overflows.
- No target specific optimizations (e.g. peephole optimizations) implemented.
- G++ frontend is partially working (e.g. no exceptions).
- Some limitations:
  - no overflow detection
  - overflows can occur in compare operations
  - ...

$\Rightarrow$ Lots of improvements are possible

Summary

- A free set of tools to develop programs for BCU 1 and BCU 2 in a RAD like way was implemented.
- A EIB bus interface daemon was developed.
- The GNU toolchain was ported to the M68HC05 architecture.
- The GCC port needed several tricks to make GCC cope with the limitations of the architecture.

Questions?

Project homepage:
`http://www.auto.tuwien.ac.at/~mkoegler/index.php/bcus`