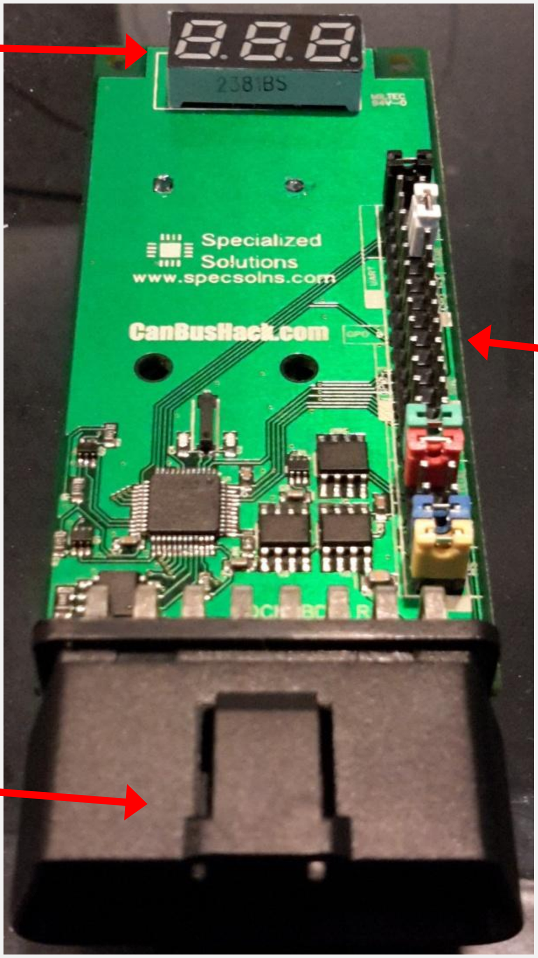
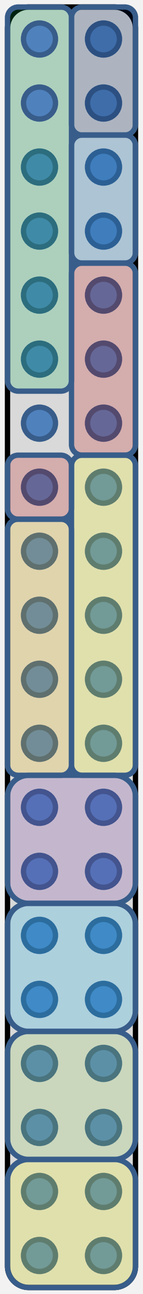
Defcon 23
Vehicle Hacking Village
The Badge and PAWN

Customizing the Badge

The Badge


Current Score
The Jumpers
OBD-II Port
Specialized Solutions
www.specsolns.com
CanBusHack.com

Badge Hardware Interfaces



Host UART for FTDI connector, GND (black) at bottom

GP Output 0

GP Inputs 0 through 3 (0 on bottom)

Selects script clear when present also is switch input

Selects GAME mode when jumper present

GP Outputs 1 through 3 (1 on bottom)

SW Debug Header, starting at top:
-3.3v
-GND
-TMS/SWDIO
-TCK/SWDCLK
-NRST

(A) Both jumpers present route CAN to single wire transceiver (used for SW CAN)

(B) Both jumpers present route CAN to DUAL wire transceiver (for HS and MS CAN)

(C) Both jumpers present select MS CAN on OBD2 (pins 3 and 11)

(D) Both jumpers present select HS CAN on OBD2 (pins 6 and 14)

Using the Other Busses

- Can select one of the following
  - HS, Dual-Wire CAN, on OBD-II pins 6 and 14
  - MS, Dual-Wire CAN, on OBD-II pins 3 and 11
  - Single-Wire CAN, for GM, on OBD-II pin 1
- For Dual-Wire, jumpers on (B)
  - Use (C) and (D) for MS/HS
- For Single-Wire, jumpers on (A)
  - (C) and (D) do not matter

What is PAWN?

- C-like, compiled but interpreted, language designed for scripting embedded targets
- Interpreted languages allow for safe, fast, efficient updates of embedded system
- Easier to write than using traditional embedded programming paradigms
- Created by Thiadmer Riemersma:
  - http://www.compuphase.com/pawn/pawn.htm
- Great documentation of the language

PAWN Intro

- Differences from C
  - Semicolons are optional
  - Basically lacks types – everything is the same size - a "cell"
  - Some new keywords
  - Supports structured data, but not in the way C does it
  - no "linking phase" – multiple scripts are pulled together using #include's
  - No function prototypes (2-pass compiler)
  - No dynamic memory in the QCM implementation

PAWN Intro Cont’d

- PAWN, by itself, is the language, the interpreter, the compiler, and a few “standard libraries” that support core operations
- The power of PAWN in the system is realized though native extensions
  - These extensions provide access to whatever the system does
- The badge has numerous native extensions
- Think of it as PAWN provides the glue to everything the badge can do
  - This frees the programmer to focus on the task at hand

What is QCM?

- The QCM acronym stands for "Quick CAN Module"
- It is a product suite of scriptable units
- The Badge is actually a mini-version of another QCM product
- The other modules add things like WiFi, multiple CAN interfaces (i.e. gateway functionality), analog I/O, better digital I/O, etc.
- The other modules are about 3 times faster and support much larger script systems

Hello World

```
main()
{
  qcm_console_enable()

  printf "Hello world\n"

  qcm_console_disable()
}
```

qcm_console Functions

- The UART port on the badge is used for programming scripts
- Can also be used for script I/O
- Scripts control the port’s destiny
- Enable() turns off programming through the port, and Disable() restores it
- No fears – The CLR/SW jumper allows to force programming mode if a script calls Enable() but does not call Disable()

The SDK

- Utilities (compiler and loader)
- Include files (native extension definitions)
- Test/Examples
- All PAWN documentation can be found on the PAWN website:
  - http://www.compuphase.com/pawn/pawn.htm

Compiling Scripts

- Use your favorite text editor to create scripts
  - Some PAWN IDE's are available but tend to focus on specific uses of PAWN (for example AMX Mod X for Half-Life engine)
  - Someday there will be a QCM-specific, lightweight, IDE
- Use a command line to compile the script
  - ..\bin\pawncc hello_world.p
- Resulting file is hello_world.amx

Programming the Badge

- Again in the console, use the QCMLoader utility
  - ..\bin\QCMLoader –c hello_world.amx
- The –c option says, load the script but keep the port open so that you can see the output from printf
- The –c option is currently only uni-directional ☹. Use Teraterm or something like it if you need bi-driectional

PAWN Event Handlers

- One of the best things (I think) about PAWN is the concept of easy event handling
- Embedded systems are often responding to stimulus.
  - Can either poll or be interrupted
  - CAN message arrived? GPIO changed? Timer expired?
- Native extensions for QCM system use these event handlers to notify scripts of events
- They are just functions in the script that are named in a special way
  - @timer0, @can_rx0, etc.
- These event handlers are enabled through other native extensions (more on this later)

Timers

- Fundamental for embedded systems
- QCM-BDG supports 5 timers
- Can be one-shot or repeating
- Steps:
  - Make the event handler
  - Call qcm_timer_start() with the desired arguments

Timers Cont’d

```
@timer0()
{
    printf(“Timer Expired\n")
    qcm_console_disable()
}

main()
{
    qcm_console_enable()
    qcm_timer_start(TIMER_0,250,false)
}
```

GPIO

- 4 output channels, 6 input channels are available to the scripts
- Outputs can be driven high or low
- Inputs can be polled or events can be setup for transitions (rising edge, falling edge, or both)
- No debouncing on purpose
- Steps
  - Create any event handlers
  - Make native calls to setup events, poll for changes, or drive outputs

GPIO Cont’d

```
@gpio_input_obd2()
{
    if (qcm_gpio_get_input(GPIO_INPUT_OBD2))
    {
        printf(“OBD2 plugged in\n”)
    }
    else
    {
        printf(“OBD2 unplugged\n”)
    }
}

main()
{
    qcm_console_enable()
    qcm_gpio_configure_handler(GPIO_INPUT_OBD2, GPIO_EVENT_ALL)
}
```

UART / Host Routines

- We've seen printf()
- Can also have event handler to process data received on UART
- 16-byte buffered input, with timer-based flush
- Steps
  - Create event handler
  - Call qcm_console_enable()

UART/Host Cont’d

```
@host_rx(data[], data_size)
{
    new i
    printf("Data Size: %d\r\nData:\r\n", data_size)

    for (i = 0; i < data_size; i++)
    {
        printf("%x ", data[i])
    }

    printf("\r\n")
}

main()
{
    qcm_console_enable()
}
```

CAN Interfaces

- Three ways to receive CAN messages:
  - Specific ID's (recommended)
  - All received messages (be careful)
  - A mix of both
- Steps
  - Create event handlers
  - Call qcm_can_init() and register any event handlers

CAN Cont’d

```
@can_rx0(rx_msg[QCM_CAN_MSG])
{
    new i

    printf("Msg Received: %x, %d, %d : ", rx_msg.id, rx_msg.is_extended, rx_msg.dlc)

    for (i = 0; i < rx_msg.dlc; i++)
    {
        printf("%x ", rx_msg.data[i])
    }

    printf("\r\n")

    rx_msg.id = 0x7E8
    qcm_can_tx(rx_msg)
}

main()
{
    qcm_can_init(500000)
    qcm_can_configure_rx_handler(CAN_RX_HANDLER_0, 0x7E0, false)
}
```

PAWN State Machines

- Very powerful PAWN built-in that eases implementations of state machines
- Basically a specific state can be tied to an event handler, and the correct event handler will be invoked, depending upon state
- So, there can be multiple event handlers that handle the same event, but the state dictates which is invoked
- Great for many embedded constructs
  - Applications
  - Parsers
  - Multi-step message handing
  - Etc.

State Machine Cont’d

```
@timer0 <pwm_low>
{
    qcm_gpio_set(GPIO_OUTPUT_0, false);
    state pwm_high
}

@timer0 <pwm_high>
{
     qcm_gpio_set(GPIO_OUTPUT_0, true);
     state pwm_low
}

main()
{
     qcm_timer_start(TIMER_0,250,true)
     state pwm_low
}
```

7-Segment Display

- Hello Word Revisited

QCM PAWN Specifics

- Floating-point is enabled along with the floating-point routines (see float example in SDK)
- Dynamic memory allocation is not available
- Most of the core library is there, with the exception of some string routines, time, and sockets
- Only the main() routine can call the “sleep” function
- Basically, latest version of PAWN with other’s contributions to work around some bugs

Putting Things Together

- Lots of examples in SDK “test” folder
- Look at the “gentle” language intro and the language guide inside the SDK or
  - http://www.compuphase.com/pawn/pawn.htm
- Thiadmer’s documentation is great and many, many thanks for creating a very powerful, but small scripting engine

Ideas

- Scripts that wait for OBD-II to be connected
- Scripts that use the 7-segment display to show OBD-II parameters
- A simple CAN diagnostic tool that uses the UART to send/receive CAN messages
- Scripts that do one-time diagnostic stuff
- Scripts that modify and resend CAN data
- Two badges linked together via UART to make a crude gateway
- Interface badge to other boards via UART/GPIO

Questions?

- See www.specsolns.com for more QCM info
- Talk to me, Nathan Hoch, about anything QCM or PAWN related
  - nhoch@specializesedsolutionsllc.com
- Other PAWN users?
- Maybe, just maybe, Thiadmer is here?
- Tweet about your Badge experiences
  - #CHVBadge