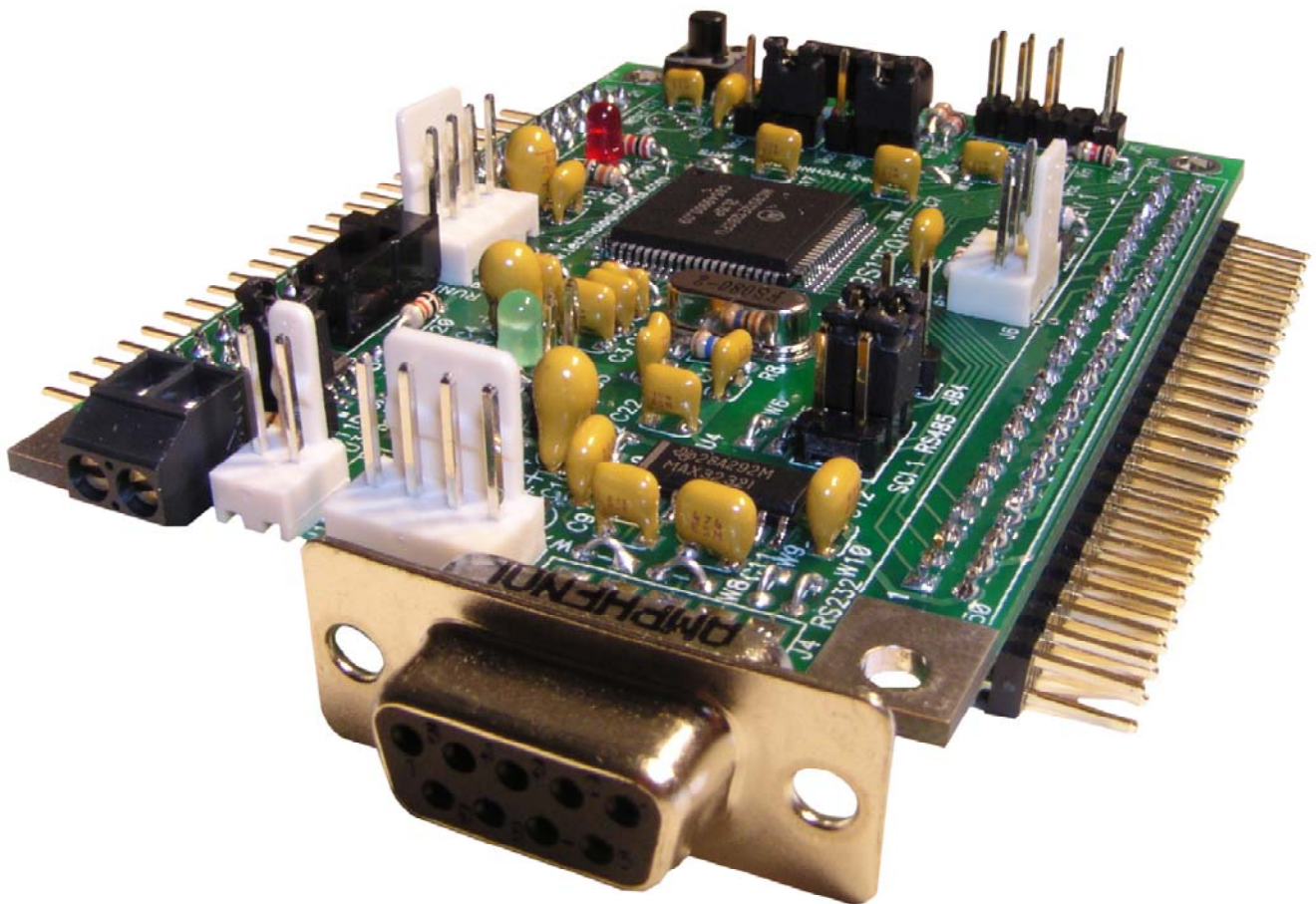


Using Your Adapt9S12E128 Microcontroller Module



www.technologicalarts.com

**For More Information On This Product,
Go to: www.freescale.com**

Revision 0

DISCLAIMER

While we have made every effort to avoid errors in the preparation of this manual, we cannot be held responsible for any misinformation or omissions that may have occurred. Furthermore, as manufacturer of this product, **Technological Arts**' sole liability and the buyer's exclusive remedy shall be refund of the amount paid or repair or replacement of the product, at the manufacturer's option. The manufacturer disclaims all other warranties, expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the product and accompanying written material, hardware, and firmware. In no event shall the manufacturer or its suppliers be held liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other loss) arising out of the use of, or inability to use, the product, even if the manufacturer has been advised of the possibility of such damages. The product is not designed, intended, nor authorized for use in applications in which the failure of the product could bring about a scenario in which personal injury or death may occur. If used in any such unintended or unauthorized application, the manufacturer and its suppliers shall be held harmless against all claims, even if any such claim alleges that the manufacturer was negligent regarding the design or implementation of the product.

Product features, availability, and prices may change without notice.

All trademarks used in this document are the property of their respective holders.

1 INTRODUCTION

1.1 CONGRATULATIONS!

With Adapt9S12E128, you are now ready to explore the power and flexibility of Motorola's most advanced 16-bit microcontroller! Whether you're new to Motorola microcontrollers or you've used some of the earlier ones, such as 68HC05, 68HC11, or 68HC12, you'll be impressed with the well thought-out design and implementation of the HCS12 family. And Adapt9S12E128 gives you the opportunity to explore the E-family's potential at a very affordable price! Add to that the proven advantages of our modular design approach, and the wide variety of available accessories, and you'll see why you picked a real winner!



1.2 SUPPORT

To help you get the most out of this product, and to make the experience as enjoyable and productive as possible, we've put together a comprehensive website, loaded with resources, support, and applications information. If you experience any difficulties, or need help with your application, the World Wide Web is arguably the most valuable resource available to you. There you'll find the latest information, software, and troubleshooting help, as well as discussion groups where you can network with people around the globe to get the answers you need. So if you still need help, or have questions after reading this manual, visit our website: www.technologicalarts.com and tap into the collective! Pay particular attention to our Support, Applications, and Resources webpages. While on our Support page, be sure to join our techart-micros online discussion group.

1.3 BOARD VARIANTS

Adapt9S12E128 was designed as an evaluation and application tool for the Motorola MCS12 "E" family of microcontrollers. Based on the 9S12E128, it is also suitable for developing applications for any of the derivatives (9S12E64 and 9S12E256). All of the subsystems are identical, with the differentiating factor among the chips being the amounts of Flash and RAM offered. Motorola offers the chips in two packages: 80-pin Quad Flat Pack (QFP) and 112-pin Thin Quad Flat Pack (TQFP). The 80-pin version has fewer I/O lines, and runs only in single-chip mode, since it lacks a memory expansion bus. Modules are available for both the 80-pin and 112-pin chip packages (Adapt9S12EQ128 and Adapt9S12E128, respectively). Unless stated otherwise, all references in this manual to Adapt9S12E128 apply equally to both the "E" and "EQ" products.

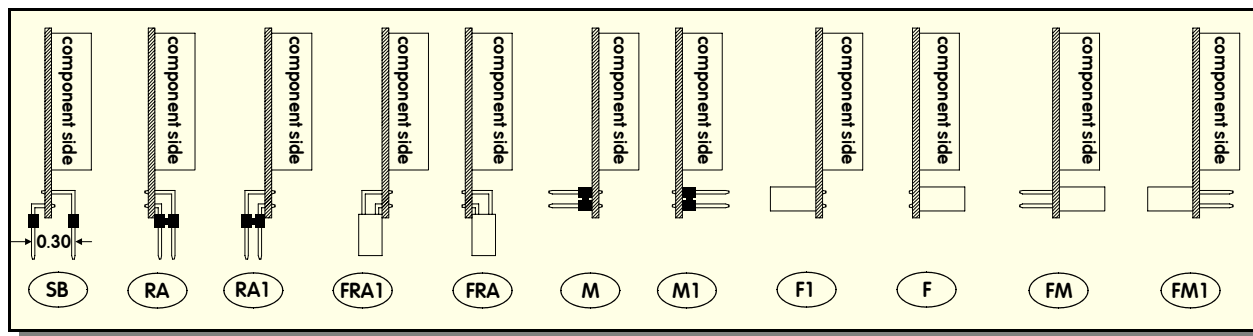
1.4 ADAPT9S12 VS. TRADITIONAL EVALUATION BOARDS

Most available evaluation and development systems tend to be too expensive and bulky for embedding into a real application, so they lie on a shelf gathering dust once you've reached a certain point in the learning curve. Or maybe you think up some clever way to hack it apart and make it fit inside your robot or product prototype. Even then, the prototyping area provided is often limited, and does not lend itself to re-usability. And what if you burn out a chip the night before the contest or product demo? What a mess to repair or re-design!

The Adapt9S12 system solves all of these problems and more! Since it brings out all I/O lines and control signals to two standard 50-pin interface connectors, it is modular and re-usable. Interface cards can be unplugged and upgraded, or the whole system can be re-configured at the last minute. And with several connector options to choose from, you can use the module in whatever way best suits your needs-- board-stacking, end-to-end (planar), backplane, ribbon cables, etc. A full range of accessories including backplanes, prototyping cards, and application-specific cards is available, and more accessories are being developed all the time. Make a point to visit our website from time to time, just to see what's new.

The connector options ("RA1") supplied in the Evaluation Package support the use of prototyping cards on each of the H1 and H2 I/O connectors, forming a planar arrangement. Advantages of this configuration are easy access of all I/O pins for probing and measurement, and easy prototyping of your

Standard Connector Options (use NC for "no connector")



interface circuits. What's more, the detachable nature of the prototyping cards means that you can easily replace them with other cards, including some of the application cards available from Technological Arts. You can build up a collection of different application circuits, and use them all with the same microcontroller board. This is especially advantageous in an educational environment, where the student can progress from simple to more complex applications throughout a semester, or from one course to the next-- perhaps even incorporating the board into a final project. In fact, where budgets are tight, different students can share the same microcontroller module, and plug in their own interface cards when it's their turn to use it.

1.5 USING ADAPT9S12E128 WITH SOLDERLESS BREADBOARDS

When used with an optional adapter (#ADHDR50-F), the module will easily plug vertically into any standard solderless breadboard. The resulting footprint is equivalent to a 50-pin narrow DIP, and has a similar pin-numbering sequence (ie. wraps around the end, from pin 25 to 26). Plug the adapter into your breadboard, wire up your circuits, as required, and then plug the module into the adapter. If you want to access signals on H2 as well, you'll need a 50-pin ribbon cable and a breadboard adapter (#ADIDC50-M) to bring those signals down to the breadboard.



Perhaps a better alternative is a planar configuration utilizing cards mounted with solderless "experimenter" cards that plug into H1 and H2 in place of the prototyping cards supplied with your package (shown). These cards (#AD12EXPH1-FRA1 and #AD12EXPH2-FRA1, respectively) feature dual-row receptacles ("F"-style connector) next to the breadboard section, giving you easy access to all of the I/O signals. To build your circuit, it's just a matter of plugging lengths of ordinary #22 hookup wire between the signals on the receptacles and the places you need them on the breadboard.



1.6 RESIDENT DEBUG/MONITOR

Residing in a 2K protected block of on-chip flash memory is Motorola's versatile Serial Monitor program. When used with our free uBug12 Windows application running on a personal computer, you can display and edit memory and registers, erase and program flash, set breakpoints, and do instruction tracing. In Run Mode, your program runs automatically from Flash, following reset. See Chapter 2 and Appendix A for details on uBug12 and the resident monitor.

1.7 EXPANDED MODE OPERATION

(Note: this mode is not available on Adapt9S12EQ128). A jumper block (JB1) is provided for setting MODA and MODB logic levels. These levels are sensed by the MCU after reset to determine the chip's operating mode (single-chip, narrow expanded, or wide expanded memory modes). You will normally run the MCU in single-chip mode, so that the monitor functions are available. In this mode, all I/O ports are available for user applications. In expanded modes, the multiplexed address and data bus lines are formed from PORTA, PORTB, and PORTK on the second 50-pin connector, H2, and are thus not available as I/O ports. When operated in this mode, additional memory can be added externally. To

make it easier for you, a low-cost RAM card is available (Order Code: AD9S12MX1). By plugging this card into H2, you can develop your programs in RAM instead of using the MCU's flash memory. You'll need to use a BDM pod, however, since the resident monitor program does not directly support external memory.

1.8 COMMUNICATIONS

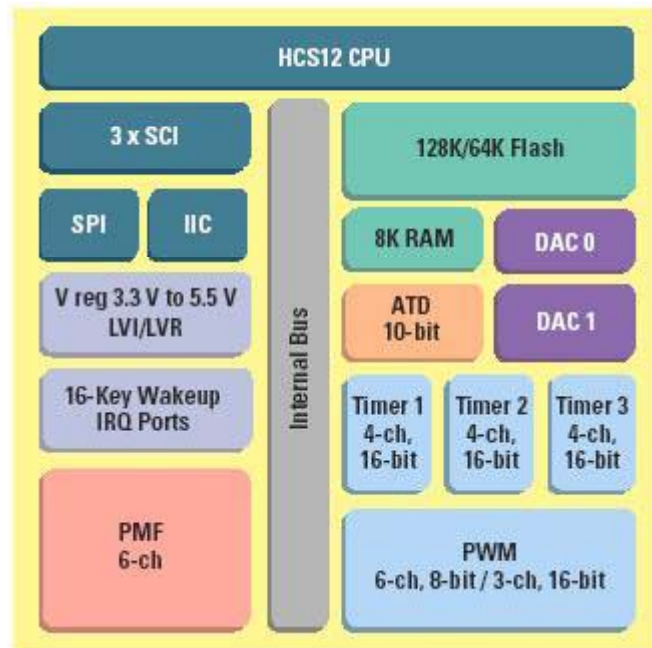
Two RS232C-compatible serial interface ports (RX & TX only) are included on Adapt9S12E128, allowing communication with a PC, or any other device which has an RS-232 serial port. The primary RS232 channel (implemented on SCI0 of the MCU) is available via a 9-pin D-sub connector. In LOAD mode, it is used to talk to the resident monitor. In RUN mode, it is available for your application. The second RS232 port (implemented via SCI1 of the MCU) is brought out to a standard 4-pin Molex connector. An optional serial cable (#SCPC9) compatible with this connector is available from Technological Arts.

An RS-485 transceiver circuit is also attached to SCI1. Use jumper block JB8 to select between the RS485 interface or the second RS232 interface, mentioned above. RS-485 is commonly used for networking industrial control applications, where long cable lengths and good noise immunity are required. For multi-drop operation, a jumperable termination resistor (R9) is provided (via W11).

An IrDA transceiver is implemented on the third SCI (SCI2). Jumper blocks JB6 and JB7 are used to select parameters for IrDA operation. Consult the schematic and the IrDA device datasheet to determine the best settings for your application.

To maximize versatility, the logic-level RXD and TXD signals from all three SCIs of the MCU are also brought out to the 50-pin headers (H1 and H2), for applications such as MIDI, or for use as general purpose I/O lines. For these applications, jumper wires on the board can be cut or removed, if necessary. Check the schematic for the relevant pins and jumper wires.

In passing, it should be mentioned that the MCU also supports SPI and IIC (also called I2C). Since these are logic-level protocols, meant for local communications among peripheral chips, no transceivers are required nor are they provided. Commonly used SPI and I2C chips and modules include serial memory, temperature controllers, clock/calendar chips, DACs, MP3 decoders, etc. See the MCU datasheets for details on these subsystems.



2 GETTING STARTED

- ◆ Perform a quick visual check of the Module for any damage during transit
- ◆ Connect the Module's RS232 port (J4) to a comport of a personal computer, using the cable supplied
- ◆ Locate and install the Windows application called uBug12, included on the CD (also on webpage)
- ◆ Launch uBug12
- ◆ Activate the serial port connection by entering CON x (where x is the comport you are using; usually 1 or 2)
- ◆ Set switch SW2 to the LOAD position
- ◆ Connect the supplied power source to J1
- ◆ uBug12 will display a short message, followed by its command prompt
- ◆ The Module is now ready for your commands (see Appendix B for full list of uBug12 commands)

To download one of the supplied example programs into flash and execute it, follow these steps:

- ◆ Type **fbulk** <enter> at the uBug12 prompt to erase any existing program
- ◆ Type **fload** <enter>
- ◆ From the displayed file browser, select one of the example program's output files (.s19 or .s28 file)
- ◆ After loading has finished, move SW2 to the RUN position, and press the Reset button (SW1)
- ◆ The program will run
- ◆ If you wish to debug the program, move SW2 back to the LOAD position and press Reset
- ◆ The uBug12 prompt will appear
- ◆ Use uBug12 commands to debug your code

2.1 POWER OPTIONS

A DC power supply is included with most bundle configurations. It is the recommended power source when you're starting out. If for some reason it's not convenient (eg. you don't want an extension cord trailing around behind your robot :-), there are a couple of alternatives:

Option 1: connect a DC voltage of 6 Volts or more (absolute maximum: 24VDC) via the external power connector, J1. Your DC supply does not need to be regulated, but it should be capable of supplying at least 100 mA (more if you will be using a BDM pod, or you're driving other circuits as well). If your supply is also driving motors, make sure to isolate it before feeding it into the module (to protect it from electrical noise generated by the motor coils). You can do this by putting inductors (10uH, nominal) in series with both the + and - leads. Preferably, use the red & black power wire provided (order code: PCJ18). Red is positive, and black is negative (ground). **CAUTION! Make sure you have the polarity correct!**

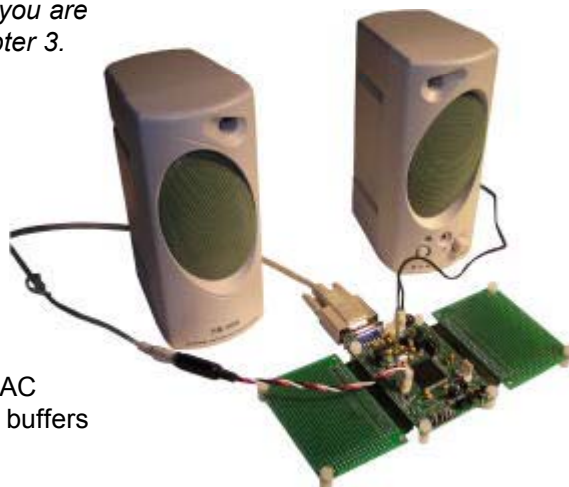
Option 2: supply regulated 5VDC (or 3V if using 3V mode) via the appropriate pins on one of the 50-pin I/O connectors (H1 or H2). Cut W4 to prevent reverse-biasing the on-board regulator. Check the schematic included with your module for the Vcc and GROUND pins on the headers. **CAUTION! Double-check your connections before applying power! If you are applying 3V, be sure to read the notes on 3-Volt operation in chapter 3.**

2.2 DEMO PROGRAMS

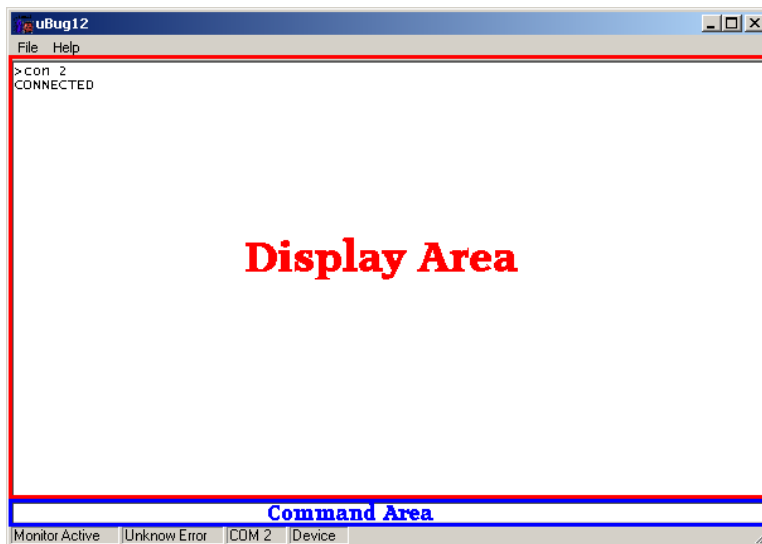
We've included a few demo programs on the CD-ROM (also available from the product webpage) to give you a starting point. There are some examples in C and some in assembler. The sourcecode is included, so you're free to modify them all you want!

2.3 THE SOUND DEMO

A sound demo program is included, which uses the two DAC channels to playback a .WAV audio file. An op amp on the board buffers



the DAC outputs so it can drive stereo headphones or amplified speakers of the sort commonly used with a personal computer. After loading the sound demo, switch to Run Mode via switch SW2. Connect the supplied serial cable between the 9-pin connector on the module To hear the audio playback emitted from the DAC, attach the included 3-wire cable between connector J6 to a pair of amplified speakers such as are commonly used with a personal computer. Set the volume control to a moderate level, and apply the included power source to your setup. You should immediately hear an audio playback on the speakers.



2.4 USING THE DEBUG/MONITOR

uBug12 is a Windows-based graphical user interface (GUI) for the Motorola's HCS12 Serial Monitor program. It aims to emulate the most common debug/monitor commands, and to provide an easy-to-use interface. The following paragraphs will help you get started with uBug12.

Install uBug12 from the included CD simply by unzipping it onto your C: drive. Launch the application, and then apply power to the module. A sign-on message will appear in the uBug12 window. A full list of uBug12 commands is provided in Appendix A.

The first step after you have launched uBug12 is to activate the connection to the target device (ie. 'Adapt9S12E128 module) which is running the Serial Monitor Program. This is done via the CON command, which takes one parameter: the number of the comport to which the device is connected.

eg. **CON 2**

In this example, uBug12 will open a connection to the target hardware attached to com2. Then you can choose any of the uBug12 commands.

One of the most useful commands is FLOAD. This command lets you load an S-record file into the target device flash. If you are loading a file containing linear S2 records, just type FLOAD, if you are using a banked S2, such as those generated by ImageCraft's ICC12, Metrowerks Codewarrior, and other tools, use FLOAD ;B. Once you have typed the FLOAD command followed by the ENTER key, a dialog box will appear and you will be able to browse to the file you want to load. If you have omitted the ;B parameter, only files with an S2 extension will be displayed for loading. Before loading another file, though, make sure to erase flash, using the FBULK command.

One nice feature you'll discover is command line history. Use the UP and DOWN arrows on your keyboard to recall previously typed commands. You can then edit them, as needed, before hitting <ENTER>.

2.5 NEW TO MOTOROLA MCUS?

If you've come from an 8051, PIC, or other background (or have never used microcontrollers before), you should get up to speed on Motorola MCUs by reading *Understanding Small Microcontrollers*, found on the CD-ROM. Written by Motorola's Jim Sibigroth, principal design engineer of the HC12 family, this excellent book uses an earlier MCU (68HC05) to introduce you to the basic concepts and design philosophy upon which the 9S12 was built. You should also make sure to have a copy of the HC11 Reference Manual, since it contains detailed descriptions and examples for many of the hardware subsystems.

2.6 MIGRATING FROM 68HC11

If you are already experienced with the 68HC11 family of microcontrollers, writing programs for the HCS12 family will not present a big challenge (don't throw away your HC11 Reference Manual-- the trusty "pink book"). In fact, you can use your existing 68HC11 assembly code and re-assemble it to run on the CPU12 core, but there are a few things to keep in mind.

Assembler syntax. You may need to edit your source file to conform to the syntax and directives requirements of the HC12 assembler you'll be using. There are several assemblers available (eg. AS12, MiniIDE, IASM12, MCUEz), and each has its own syntax to be aware of.

Register Block. Instead of \$1000, the register block default location is \$0000 through \$03FF, and there are a few hundred registers! You'll need to locate the relevant registers for the subsystems you plan to use, and make sure they are properly configured.

RAM location. The RAM overlaps the register block area, starting at \$0000, with register taking priority, so the first 1K bytes of RAM are not usable. The monitor program re-maps RAM to start at \$2000 and go to \$3FFF. This means you would initialize the Stack Pointer to \$4000 (on the HCS12, the stack pointer points to the address following the top of the stack).

High-speed Bus. The default bus speed is half the crystal frequency of 8 MHz, so it is 4 MHz. If you enable the PLL, it will be even higher (up to 24 MHz). This will mean changing some initialization values for control registers and revising delay constants if you are using any software timing loops.

I/O Ports. The digital I/O ports on the HCS12 are more flexible than ever. Besides selecting the direction of each port pin via a Data Direction Register, there are registers controlling output drive level (standard and reduced), internal pullup and pulldown resistors, and output logic polarity (ie. true or inverted logic).

COP Watchdog. On most flavours of HC11, this was enabled via a bit in the non-volatile CONFIG register. On the HC12, it is dynamic, and automatically enabled following reset. Therefore you have to choose whether you're going to service it, or disable it.

Write-Once Registers. On the HCS12, there is no 64-cycle startup window in which you have to write all the protected registers. Instead, the HC12 implements a WriteOnce rule on sensitive registers. What this means is that, following reset, you have one chance to write them, then they become "Read Only". The advantage of this is that you have more control of when you alter these register values. To take advantage of this safeguard, you should initialize all the registers that are crucial, even if the default values are what you want. That way, if your code runs amok, or there are any glitches which try to change register values, they will be protected.

There are many more differences, and you should make sure to read through the Motorola App Note (AN1284) on the CD-ROM that details the new instructions and addressing modes of the 68HC12, explaining differences from the 68HC11. The book Programming the Motorola 68HC12 Family (#PMM929, available from Technological Arts) thoroughly covers all the subsystems of the 68HC12, providing highlights of the differences with respect to 68HC11, and is recommended reading.

2.7 MIGRATING FROM THE 68HC912

You gain a lot more speed, memory, and flexibility, but you have a lot more registers to think about, and many of their addresses have changed. Gone are the Vfp generator and flash voltage switch, since the new flash technology uses 5V, and has built-in self-timed algorithms for program and erase functions. But your s-records must contain an even number of bytes, and begin on an even address boundary if you're going to "burn" them into flash. Some assemblers will generate this format for you but others, such as the included AS12, don't.

3 HARDWARE DESIGN FEATURES

3.1 3-VOLT OPERATION

One of the nice features of the MCU is that it can operate on 3 to 5 V, while maintaining full bus speed capability. To support 3 Volt operation, the module incorporates an adjustable regulator whose output voltage is set by a resistive voltage divider. The circuit has been designed such that simply inserting a shorting jumper causes the regulator's output to shift from 5V to 3.3V. When operated at 3V, there are a few precautions that should be noted, however.

The logic pins are not 5V-tolerant, so you will need to take the necessary steps to prevent damage to the I/O pins of the MCU. Also, the maximum VRH voltage is limited to 3.3V, so any external voltage or precision voltage reference you supply should be scaled accordingly. One last point is that some BDM pods (eg. MicroBDM12SX) will not work with 3V targets, so you should check the specs of the BDM pod you intend to use.

3.2 RESET

Unlike previous HC11 and HC12 designs, the 9S12E MCU has an on-chip low-voltage inhibit (LVI) reset circuit, so it is not necessary to provide such a circuit externally. A momentary tact switch is provided for manual reset, and the LVI circuit will provide a clean reset signal upon power-up.

3.3 USING A PRECISION VOLTAGE REFERENCE

Adapt9S12E128 uses Vcc as the voltage reference (VRH) for the analog-to-digital converter on the MCU. If you wish to use a precision voltage reference, you may insert an LM385 or LM336 or equivalent part in location U2. Make sure it is a TO-92 style package. Remove jumper W11 only if you wish to use a VRL other than system ground. It is easiest to use a 2.5V reference, but you'll have to make sure none of your analog input voltages will exceed 2.5V, or you risk burning out the analog input circuits. If you use a 5V reference chip, you'll need to supply it with more than 5V. To supply it from Vin (from J1), move link JB9 to the pin1-pin2 location. Refer to the board schematic if you're unclear on what to do. If you are operating the board in 3Volt mode, make sure not to exceed the 3V limit on VRH. Consult the MCU datasheet to ensure that you do not exceed the manufacturer's limits on the voltage reference pins.

3.4 ABOUT THE ON-BOARD VOLTAGE REGULATOR

Adapt9S12E128 includes an LM1086CT-ADJ voltage regulator. Housed in a TO-220 package, it is capable of dissipating about 500 mW at room temperature. Other nice features are: reasonably low quiescent current, and low dropout voltage-- it will work with an input voltage down to about 5 Volts, making it quite well-suited to battery operation. It is also designed to withstand reverse polarity. One drawback, however, is that it can become unstable and start to oscillate at low temperatures, especially if the input voltage source is connected to J1 via long wires. If low-temperature operation is anticipated, the on-board 10uF tantalum capacitor can be replaced with a higher value (47uF or 100uF). To compensate for long lead-in wires, add capacitance of 100uF or so at, or close to, the J1 connector. A couple of pads are provided on the circuit board to accommodate this.

Heatsinking. Because the regulator is mounted on the underside of the circuit board, with the package body parallel to the plane of the board, it can be safely attached to a heatsink, if desired. Many clip-on heatsinks are available for use with TO-220 packages. Another option is to mount the board on a sheet of aluminum, using standoffs. If the appropriate length is chosen for the standoffs, the tab of the regulator will lie flush with the aluminum sheet, and can be coated with silicone grease and bolted (or riveted) to the plate (tightening a nut and bolt will require a little ingenuity). Note that the metal tab of the regulator is not Ground (it is Vin), so in most cases you will need a heatsink insulation kit.

3.5 DAC

While the on-chip two-channel DAC pins are brought out to I/O connector H1, the signals are also routed through an op amp buffer circuit so that they can be used with headphones or amplified

speakers in audio applications.

3.6 IIC

The MCU includes an inter-integrated circuit interface, commonly referred to as IIC or I2C. Connector J7 brings out SCL, SDA, Vcc, and Ground for easy access to this interface..

3.7 PLL

While the supplied crystal is only 8MHz, the MCU is capable of running at a much higher speed. The phase-locked loop feature of the MCU allows you to boost the bus speed by an integer multiple of the crystal frequency, so by enabling the PLL, you can actually run the MCU at 24MHz.

3.8 IRDA

A popular wireless interface on PDAs and some personal computers is IrDA. The MCU supports the variable bit-length protocol required by IrDA, so an IrDA physical interface has been included on the module. The device used is made by Telefunken, and the manufacturer's data sheet is included on the CD. Consult the data sheet for details on how to use the IR sensitivity and Shutdown features of this device. As of this writing, Motorola is preparing an application note and software example for an IrDA stack for the 'E128. Check their website for news on this.

3.9 D-BUG12 MODE SELECT

Some previous HCS12 chips had versions of Motorola's D-Bug12 debug/monitor program available for them. As of this writing, D-Bug12 is not available for the "E" series chips. Since the 2K serial monitor/debugger is included on-chip, there is no need for D-Bug12, and it is likely not going to be implemented. The Mode Select jumper block was implemented on the circuit board anyway, in case some customers wish to implement their own monitor program.

4 WRITING SOFTWARE

4.1 IMPACT OF THE SERIAL MONITOR

When you are working without a BDM pod, the Serial Monitor program is the only method available to load and erase flash. It is in a protected block of flash, so there's no way to accidentally erase it. There are two modes, controlled by switch SW2: Run and Load. The monitor mode is determined immediately following reset by checking the position of switch SW2. When working with the monitor program in place, there are a few points to be noted:

1) while the user vectors are implemented by the monitor at 0xF780 to 0xF7FF, you don't really have to worry about it, because the monitor program will automatically adjust them when your s-record is loaded.

2) the monitor relocates RAM to the address range 0x2000 to 0x4000 from the default location after MCU reset of 0x0000 to 0x1FFF.

3) the monitor program enables the phase-locked loop (PLL), so the target is running at 24Mhz (when in LOAD mode) and not at the startup speed of 4Mhz.

4) the user code must clear the CCR I-Bit, either via a CLI in assembler or via the INTR_ON() in ICC12.

5) SCI0 cannot be used by the user program when in LOAD mode, since it is dedicated to the monitor program; however, SCI1 is available to the user

6) COP cannot be disabled in Load mode.

4.2 WRITING A SIMPLE C PROGRAM IN ICC12

Before starting, you'll need to set up your compiler settings, as follows:

Program Memory = 0x4000.0x7FFF:0xC000.0xFFFF

Data Memory = 0x2000

Stack Pointer = 0x3FC0

Note that the Data Memory and Stack Pointer addresses shown are valid only for a device with a resident monitor, since the monitor remaps the RAM following reset. If you are writing software for a completely blank chip, and loading it in via a BDM pod, you'll need to change these values to work with the default RAM address range (see the MCU datasheet).

```
//this program flashes LED D1 on PP0 twice a second
#include <hcs12e128.h>

#define DUMMY_ENTRY (void (*)(void))0xFFFF

#pragma nonpaged_function _startextern void _start(void); /* entry point in crt12.s */
void main(){
    INTR_ON();           //needed for the SerialMonitor

    DDRP = 0x01; //Enable LED port

    RTICTL = 0x7F;       //Set RTI divider for 4Hz time base
    CRGFLG |= 0x80;      //Clear the RTI Flag
    CRGINT |= 0x80;      //Enable the RTI
}
```

```
#pragma interrupt_handler rti_handler
void rti_handler(){
    CRGFLG |= 0x80; // Clear the RTI Flag
    PTP ^= 0x01; //Toggle LED
    INTR_ON(); //Enable Interrupts
}

#pragma abs_address:0xFFFF0

void (*interrupt_vectors[])(void) =
{
    rti_handler, /*Real Time Interrupt*/
    DUMMY_ENTRY, /*IRQ*/
    DUMMY_ENTRY, /*XIRQ*/
    DUMMY_ENTRY, /*SWI*/
    DUMMY_ENTRY, /*Unimplement Intruction Trap*/
    DUMMY_ENTRY, /*COP failure reset*/
    DUMMY_ENTRY, /*Clock monitor fail reset*/
    _start, /*Reset*/
};
```

4.3 OTHER E128 ISSUES WITH ICC12

Because the register addresses have changed from what they were in HC12, meaning the header file is different for the E128, some library files in ICC12 will need to be re-compiled, using the new header file, if you want to use them. Of course, if you're not using library functions, or you are using functions that don't involve registers, then there won't be a problem with the existing versions. The modified functions are included on the CD-ROM to get you started.

To use the SCI, make sure to include **E128_iochar.c** and **E128_serial.c**. Also, you'll need the complete vector file for the E128, which is called **E128_vectors.c**. Unzip **E128_C.zip** and place **hcs12e128.h** in **c:\vicc\include** (or in your equivalent path). Make sure to place **E128_Vectors.c** in the same folder as your project, and add it to you project via the "add file menu item".

4.4 "HELLO WORLD" PROGRAM

First of all create a new project from the Project menu.

Then create a new file and save it as HelloWorld.c Add it to the Project by right clicking in the Project Panel and using Add Files to add it to the Files section.

Next type in the following code:

```
#define _SCI
#include <hcs12e128.h>

#pragma nonpaged_function _start
extern void _start(void); /* entry point in crt12.s */

extern int _textmode;

int putchar(char c)
{
    if (_textmode && c == '\n')
        putchar('\r');
    while ((SC0SR1 & TDRE) == 0)
        ;
    SC0DRL = c;
```

```

    return c;
}

void main(){
    INTR_ON();           //need for the SerialMonitor

    DDRP = 0x01;  //Enable LED
    SCIOBD = 26; //9600 Baud
    SCIOCR2 = 0x0C; /* enable transmitter and receiver */

    puts("Hello, World!");

}

#pragma abs_address:0xFFFFE

void (*interrupt_vectors[])(void) =
{
    _start, /*Reset*/
};

```

Since **puts** calls **putchar**, we define it before invoking it in **main**. **Main** has an implicit **_Start** entry point, which is called after the setup by **CRT12.o**, which is a module that the ICC12 linker links in as the starting point of the program. Besides initializing the stack and other system features it initializes memory, initialized variables and constants before transferring control to the Main.

Compile and link the program, fixing any syntax errors that may have cropped up. Ensure that the Project Options | Device Configuration drop down box points to the 9S12E128 Flash Mode. This sets the link address to start the code section at 0x4000 and the stack at top of RAM (0x4000).

4.5 USING A BDM POD

If you have a BDM pod, you can erase the resident monitor program completely. This will free up all the MCU resources for your program (most importantly, the SCIs). Without the monitor in place, the RAM will be at the default location following reset, so make sure to use the correct compiler/linker settings. Also, the PLL won't be enabled, so the bus speed will be 4 MHz.

4.6 AUTOMATING S-RECORD CONVERSION IN ICC12

You may have to convert the s-record file to get it into the proper format for your BDM pod to load correctly. ICC12 has a nice feature at *Project->Options->Compiler->ExecuteCommandAfterBuild* where you can add the srecvt command mentioned earlier

5 GOING FURTHER

If you'd like to get started interfacing common electronic devices such as LEDs, switches, relays, etc., you may consider purchasing the optional Demo Card (Adapt9S12DemoH1, shown). It includes a light sensor, thermistor, and a couple of logic MOSFETs.

If you're interested in more sophisticated software debugging, you may wish to consider adding a RAM expansion card (Adapt9S12MX1, shown). Together with a BDM pod, it will enable you to use third-party source-level debuggers for more productive code development.

For robotics applications, you may wish to consider adding a Servo/Sensor Interface Module (Adapt9S12SSIM, shown) to interface to popular hobby servos, distance-measuring sensors, and add sound output and sensing capabilities. The SSIM plugs onto H1 of the MCU module.

Embedded web servers and internet connectivity for microcontrollers are getting a lot of attention these days. If you are interested in exploring such applications, you may be interested in a 100-BaseT LAN card (Adapt9S12LAN, shown) that works with Adapt9S12E128 running in expanded mode.



APPENDIX A - SERIAL MONITOR

INTRODUCTION

This appendix describes the Motorola 2 Kbyte monitor program for the HC9S12 series MCU. This program supports 23 primitive debug commands to allow FLASH / EEPROM programming and debug through an RS232 serial interface to a personal computer. These include commands to reset the target MCU, read or modify memory (including FLASH /EEPROM memory), read or modify CPU registers, go, halt, or trace single instructions. In order to allow a user to specify the address of each interrupt service routine, this monitor redirects interrupt vectors to an unprotected portion of FLASH just below the protected monitor program. This monitor is intended to be device unspecific, this single application with very slight modification should execute on any HC9S12 derivative. A user on a tight budget can evaluate the MCU by writing programs, programming them into the MCU, then debug using only a serial I/O cable and free software (uBug12) for their personal computer.

This monitor does not use any RAM other than the stack itself. The COP watchdog is utilized for a cold reset function; user code should not disable the COP (ie. by writing 0x00 to COPCTL). This development environment assumes you reset to the monitor when you are going to perform debug operations. If your code takes control directly from reset, and then an SCI0 interrupt or a SWI attempts to enter the monitor, the monitor may not function because SCI0, the phase locked loop (PLL), and memory initialization registers may not be initialized as they would be for a cold reset into the monitor. There is no error handling for the PLL. If the frequency source is missing or broken, the monitor will not function. The monitor sets the operating speed of the MCU to 24 MHz. Modification of the MCU speed by the user with out considerations for the monitor program will render the monitor nonfunctional. If the PLL loses lock during operation, the monitor will fail.

BLOCK PROTECTION

In order to prevent accidental changes to the monitor program itself, the 2 Kbyte block of FLASH memory where it resides (\$F800-\$FFFF), is block protected. Additionally all write commands are restricted from modifying the monitor memory space. The only way to change the contents of this protected block is to use a BDM-based development. In the lowest cost applications where the monitor is used with an SCI serial interface to the RS232 serial port on a personal computer, there is no way to accidentally erase or modify the monitor software.

COP CONFIGURATION

The monitor as written creates hard reset function by using the COP watchdog timer. It does so by enabling the COP and waiting for a COP timeout reset to occur. If the user application uses the COP two issues must be considered.

- If the COP is disabled in the user application, the monitor will be unable to perform a hard reset and will soft reset to the start of the monitor instead.
- The monitor does not service the COP timer. If the user application implements COP timer servicing, upon re-entry into the monitor a hard reset is likely to occur.

MEMORY CONFIGURATION

- 1) Register space is \$0000-\$03FF.
- 2) Flash memory is any address greater than \$4000. All paged addresses are assumed to be Flash memory.
- 3) RAM ends at \$3FFF and builds down to the limit of the device's available RAM.
- 4) External devices attached to the multiplexed external bus interface are not supported.

SERIAL PORT USAGE

In order for this monitor to function the SCI0 serial interface is used. It is assumed that the monitor has exclusive use of this interface. User application code should not implement communications on this serial channel. This monitor accommodates RS232 serial communications through SCI0 at 115.2 kbaud.

For applications requiring the use of SCI0, you should purchase a BDM pod which allows for more advanced debugging.

VECTOR REDIRECTION AND INTERRUPT USE

Access to the user vectors is accomplished via a jump table located within the monitor memory space. This table points all interrupt sources to a duplicate vector table located just below the monitor. (\$F780-\$F7FE). The monitor will automatically redirect vector programming operations to these user vectors. The user's code should therefore continue to implement the normal (non-monitor) vector locations (\$FF80-\$FFFE). If execution of an interrupt with an un-programmed vector is attempted, behavior is undefined. For this reason, the user is strongly encouraged to implement a software trace for all vectors, as is good programming practice. The monitor depends on interrupts being available for monitor re-entry after GO or TRACE commands. Therefore, it is important that the user application executes with interrupts enabled.

APPENDIX B - UBUG12 COMMAND LIST

REGISTERS

RD		- Register Display
RM	<RegisterName> <Data8/16>	- Register Modify
CCR	<Data8>	- Set CCR register
D	<Data16>	- Set D register
PC	<Data16>	- Set PC register
PP	<Data8>	- Set PP register
SP	<Data16>	- Set SP register
X	<Data16>	- Set X register
Y	<Data16>	- Set Y register

MEMORY MODIFY

BF	<StartAdd> <EndAdd> <Data8>	- Block fill
BFW	<StartAdd> <EndAdd> <Data16>	- Block fill word
MD	<StartAdd> [<EndAdd>]	- Memory display
MDW	<StartAdd> [<EndAdd>]	- Memory display word
MM	<Address> <Data8>	- Memory modify byte
MMW	<Address> <Data16>	- Memory modify word

FLASH

FBULK		- Flash bulk erase
FLOAD	[:B][:M]	- Flash load

DEVICE INFO

DEVICE		- Get device name
--------	--	-------------------

GO/HALT

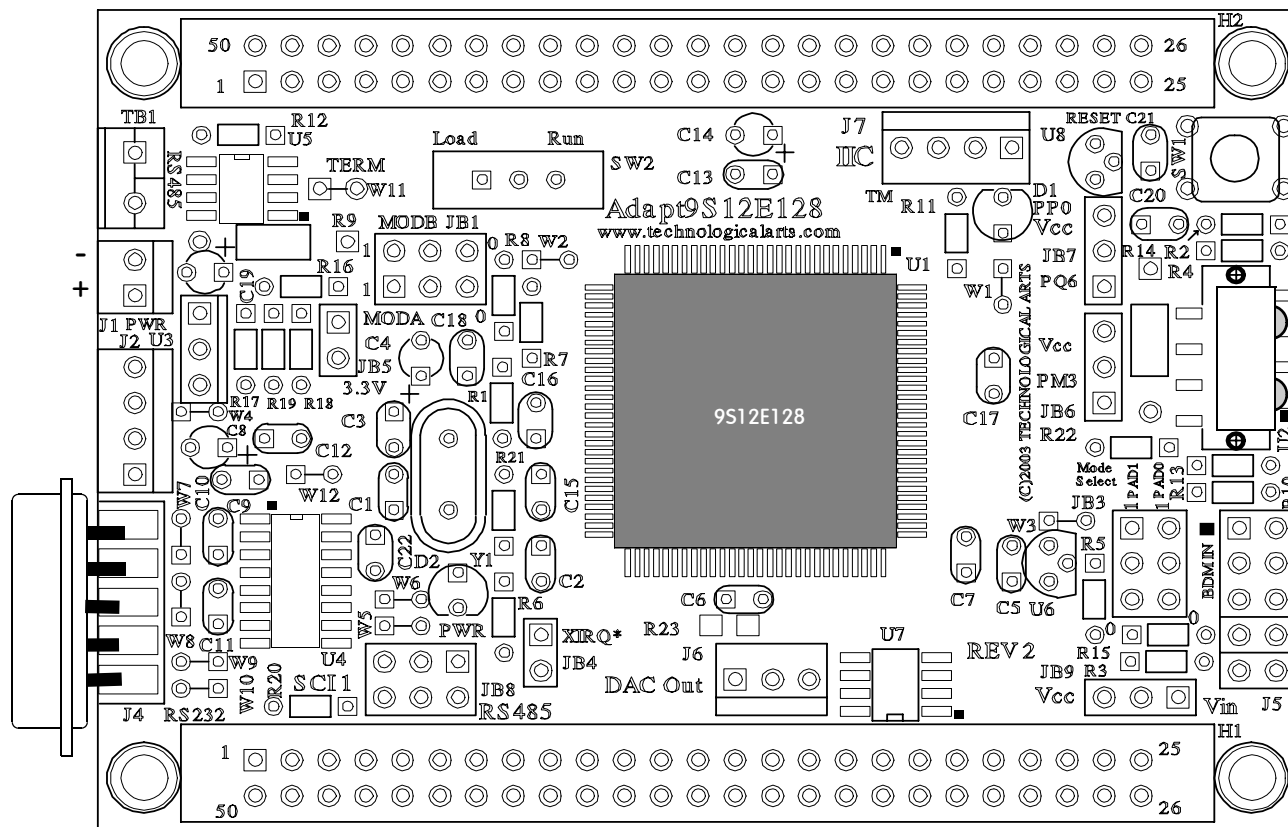
GO	[<StartAddress>]	- Start execution
HALT		- Halt execution
RESET		- Reset target

GUI

CON	<Comport>	- Connect to target
DISCON		- Disconnect from target
EXIT		- Terminate GUI
HELP		- Display help
OP	<Opacity%>	- Set main GUI opacity

APPENDIX C - COMPONENT PLACEMENT AND PINOUT

H2 - Secondary I/O or Memory Expansion



H1 - Primary I/O

H1				H2			
PIN	SIGNAL NAME	PIN	SIGNAL NAME	PIN	SIGNAL NAME	PIN	SIGNAL NAME
1	PS4/MISO	50	GROUND	1	PA7/ADDR15/DATA15	50	VCC (+5VDC)
2	PS5/MOSI	49	GROUND	2	PA6/ADDR14/DATA14	49	GROUND
3	PS6/SCK	48	PS0/RXD0	3	PA5/ADDR13/DATA13	48	PK7/ECS*
4	PS7/SS*	47	+5VDC	4	PA4/ADDR12/DATA12	47	PK6/XCS*
5	PS1/TXD0	46	PE1/IRQ*	5	PA3/ADDR11/DATA11	46	PK5/XADDR19
6	PT7/IOC17	45	PE0/XIRQ*	6	PA2/ADDR10/DATA10	45	PK4/XADDR18
7	PT6/IOC16	44	RESET*	7	PA1/ADDR9/DATA9	44	PK3/XADDR17
8	PT5/IOC15	43	PE7/NOACC/XCLKS*	8	PA0/ADDR8/DATA8	43	PK2/XADDR16
9	PT4/IOC14	42	PU0/PWM10/IOC24	9	PB7/ADDR7/DATA7	42	PK1/XADDR15
10	PT3/IOC07	41	PU1/PWM11/IOC25	10	PB6/ADDR6/DATA6	41	PK0/XADDR14
11	PT2/IOC06	40	PU2/PWM12/IOC26	11	PB5/ADDR5/DATA5	40	PQ0/FAULT0
12	PT1/IOC05	39	PU3/PWM13/IOC27	12	PB4/ADDR4/DATA4	39	PQ1/FAULT1
13	PT0/IOC04	38	PU4/PWM14	13	PB3/ADDR3/DATA3	38	PQ2/FAULT2
14	PM0/DAC0	37	PU5/PWM15	14	PB2/ADDR2/DATA2	37	PQ3/FAULT3
15	PM1/DAC1	36	PU6	15	PB1/ADDR1/DATA1	36	PQ4/IS0*
16	PP5/PMF5	35	PU7	16	PB0/ADDR0/DATA0	35	PQ5/IS1*
17	PP4/PMF4	34	PS2/RXD1	17	R/W*/PE2	34	PQ6/IS2*
18	PP3/PMF3	33	PE4/ECLK	18	ECLK/PE4	33	PM3
19	PP2/PMF2	32	PS3/TXD1	19	LSTRB*/PE3	32	PM4/RXD2
20	PP1/PMF1	31	VRL	20	IRQ*/PE1	31	PM5/TXD2
21	PP0/PMF0	30	VRH	21	PM7/SCL	30	PM6/SDA
22	PAD0/KWAD0	29	PAD4/KWAD4	22	PAD8/KWAD8	29	PAD12/KWAD12
23	PAD1/KWAD1	28	PAD5/KWAD5	23	PAD9/KWAD9	28	PAD13/KWAD13
24	PAD2/KWAD2	27	PAD6/KWAD6	24	PAD10/KWAD10	27	PAD14/KWAD14
25	PAD3/KWAD3	26	PAD7/KWAD7	25	PAD11/KWAD11	26	PAD15/KWAD15

NOTES: * indicates active low signal