Get Better Code Density than 8/16 bit MCU’s
NXP LPC1100 Cortex M0

Oct 2009
Outline

- Introduction

- ARM Cortex-M0 processor

- Why processor bit width doesn’t matter
  - Code size
  - Performance
  - Cost

- Conclusions
**ARM Cortex-M Processors**

- **Cortex-M family optimised for deeply embedded**
  - Microcontroller and low-power applications

---

**ARM Cortex-A Series:**
Applications processors for feature-rich OS and user applications

**ARM Cortex-R Series:**
Embedded processors for real-time signal processing and control applications

**ARM Cortex-M Series:**
Deeply embedded processors optimized for microcontroller and low-power applications
ARM Cortex-M0 Processor

- **32-bit ARM RISC processor**
  - Thumb 16-bit instruction set

- **Very power and area optimized**
  - Designed for low cost, low power

- **Automatic state saving on interrupts and exceptions**
  - Low software overhead on exception entry and exit

- **Deterministic instruction execution timing**
  - Instructions always takes the same time to execute*

*Assumes deterministic memory system
Thumb instruction set

- **32-bit operations, 16-bit instructions**
  - Introduced in ARM7TDMI (‘T’ stands for Thumb)
  - Supported in every ARM processor developed since
  - Smaller code footprint

- **Thumb-2**
  - All processor operations can all be handled in ‘Thumb’ state
  - Enables a performance optimised blend of 16/32-bit instructions
  - Supported in all Cortex processors
Instruction set architecture

- Based on 16-bit Thumb ISA from ARM7TDMI
  - Just 56 instructions, all with guaranteed execution time
  - 8, 16 or 32-bit data transfers possible in one instruction

### Thumb
User assembly code, compiler generated

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADD</th>
<th>ADR</th>
<th>AND</th>
<th>ASR</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIC</td>
<td>BL</td>
<td>BX</td>
<td>CMN</td>
<td>CMP</td>
<td></td>
</tr>
<tr>
<td>EOR</td>
<td>LDM</td>
<td>LDR</td>
<td>LDRB</td>
<td>LDRH</td>
<td>LDRSB</td>
</tr>
<tr>
<td>LDRSH</td>
<td>LSL</td>
<td>LSR</td>
<td>MOV</td>
<td>MUL</td>
<td>MVN</td>
</tr>
<tr>
<td>ORR</td>
<td>POP</td>
<td>PUSH</td>
<td>ROR</td>
<td>RSB</td>
<td>SBC</td>
</tr>
<tr>
<td>STM</td>
<td>STR</td>
<td>STRB</td>
<td>STRH</td>
<td>SUB</td>
<td>SVC</td>
</tr>
<tr>
<td>TST</td>
<td>BKPT</td>
<td>BLX</td>
<td>CPS</td>
<td>REV</td>
<td>REV16</td>
</tr>
<tr>
<td>REVSH</td>
<td>SXTB</td>
<td>SXTH</td>
<td>UXTB</td>
<td>UXTH</td>
<td></td>
</tr>
</tbody>
</table>

### Thumb-2
System, OS

- NOP
- SEV
- WFI
- WFE
- YIELD
- DMB
- DSB
- ISB
- MRS
- MSR
Program registers

- All registers are 32-bit wide
  - Instructions exist to support 8/16/32-bit data

- 13 general purpose registers
  - Registers r0 – r7 (Low registers)
  - Registers r8 – r12 (High registers)

- 3 registers with special meaning/usage
  - Stack Pointer (SP) – r13
  - Link Register (LR) – r14
  - Program Counter (PC) – r15

- Special-purpose registers - xPSR
Instruction behaviour

- Most instructions occupy 2 bytes of memory

\[
a = a \times b; \quad \text{C code}
\]
\[
\text{MUL r0, r1;} \quad \text{Assembler}
\]

- When executed, complete in a fixed time
  - Data processing (e.g. add, shift, logical OR) take 1 cycle
  - Data transfers (e.g. load, store) take 2 cycles
  - Branches, when taken, take 3 cycles

- The instructions operate on 32-bit data values
  - Processor registers and ALU are 32-bit wide!
Thumb instructions

- Cortex M0 requires instruction fetches to be half word aligned
- Thumb instructions are aligned on a two-byte boundaries

32 bit instructions are organized as 2 half words
Nested Vectored Interrupt Controller

- **NVIC enables efficient exception handling**
  - Integrated within the processor - closely coupled with the core
  - Handles system exceptions & interrupts

- **The NVIC includes support for**
  - Prioritization of exceptions
  - Tail-chaining & Late arriving interrupts

- **Fully deterministic exception handling timing behavior**
  - Always takes the same number of cycles to handle an exception
  - Fixed at 16 clocks for no jitter
  - Register to trade off latency versus jitter

- **Everything can be written in C**
Interrupt behaviour

- On interrupt, hardware automatically stacks corruptible state
- Interrupt handlers can be written fully in C
  - Stack content supports C/C++ ARM Architecture Procedure Calling Standard
- Processor fetches initial stack pointer from 0x0 on reset
Writing interrupt handlers

Traditional approach

- **Exception table**
  - Fetch instruction to branch

- **Top-level handler**
  - Routine handles re-entrancy

```assembly
IRQVECTOR
  LDR   PC, IRQHandler

IRQHandler PROC
  STMFD sp!,{r0-r4,r12,lr}
  MOV   r4,#0x80000000
  LDR   r0,[r4,#0]
  SUB    sp,sp,#4
  CMP    r0,#1
  BNE    C_int_handler
  MOV   r0,#0
  STR    r0,[r4,#4]
  ADD    sp,sp,#4
  LDMFD sp!,{r0-r4,r12,lr}
  SUBS   pc,lr,#4
ENDP
```

ARM Cortex-M family

- **NVIC automatically handles**
  - Saving corruptible registers
  - Exception prioritization
  - Exception nesting

- **ISR can be written directly in C**
  - Pointer to C routine at vector
  - ISR is a C function

- **Faster interrupt response**
  - With less software effort

- **WFI, sleep on exit**
Software support for sleep modes

- ARM Cortex-M family has architected support for sleep states
  - Enables ultra low-power standby operation
  - Critical for extended life battery based applications
  - Includes very low gate count Wake-Up Interrupt Controller (WIC)

- **Sleep**
  - CPU can be clock gated
  - NVIC remains sensitive to interrupts

- **Deep sleep**
  - WIC remains sensitive to selected interrupts
  - Cortex-M0 can be put into state retention

- **WIC signals wake-up to PMU**
  - Core can be woken almost instantaneously
  - React to critical external events
# Instruction set comparison

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADD</th>
<th>ADR</th>
<th>AND</th>
<th>ASR</th>
<th>B</th>
<th>CLZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFC</td>
<td>BFI</td>
<td>BIC</td>
<td>CDP</td>
<td>CLREX</td>
<td>CBNZ</td>
<td>CBZ</td>
</tr>
<tr>
<td>CMP</td>
<td></td>
<td></td>
<td></td>
<td>DBG</td>
<td>EOR</td>
<td>LDC</td>
</tr>
<tr>
<td>LDMIA</td>
<td></td>
<td></td>
<td></td>
<td>LDMDB</td>
<td>LDR</td>
<td>LDRB</td>
</tr>
<tr>
<td>LDRBT</td>
<td></td>
<td></td>
<td></td>
<td>LDRD</td>
<td>LDREK</td>
<td>LDREKB</td>
</tr>
<tr>
<td>LDREKH</td>
<td></td>
<td></td>
<td></td>
<td>LDRH</td>
<td>LDRHT</td>
<td>LDRSB</td>
</tr>
<tr>
<td>LDRSBT</td>
<td></td>
<td></td>
<td></td>
<td>LDRSHT</td>
<td>LDRSH</td>
<td>LDRT</td>
</tr>
<tr>
<td>MCR</td>
<td></td>
<td></td>
<td></td>
<td>LSL</td>
<td>LSR</td>
<td>MLS</td>
</tr>
<tr>
<td>MCRR</td>
<td></td>
<td></td>
<td></td>
<td>MLA</td>
<td>MOV</td>
<td>MOVT</td>
</tr>
<tr>
<td>MRC</td>
<td></td>
<td></td>
<td></td>
<td>MRRC</td>
<td>MUL</td>
<td>MVN</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td>ORN</td>
<td>ORR</td>
<td>PLD</td>
</tr>
<tr>
<td>PLDW</td>
<td></td>
<td></td>
<td></td>
<td>PLI</td>
<td>POP</td>
<td>PUSH</td>
</tr>
<tr>
<td>RBIT</td>
<td></td>
<td></td>
<td></td>
<td>REV</td>
<td>REV16</td>
<td>REVSH</td>
</tr>
<tr>
<td>ROR</td>
<td></td>
<td></td>
<td></td>
<td>RRS</td>
<td>RSB</td>
<td>SBC</td>
</tr>
<tr>
<td>SBFX</td>
<td></td>
<td></td>
<td></td>
<td>SDIV</td>
<td>SEV</td>
<td>SMLAL</td>
</tr>
<tr>
<td>SMULL</td>
<td></td>
<td></td>
<td></td>
<td>SSAT</td>
<td>STC</td>
<td>STMIA</td>
</tr>
<tr>
<td>STMDB</td>
<td></td>
<td></td>
<td></td>
<td>STR</td>
<td>STRB</td>
<td>STRBT</td>
</tr>
<tr>
<td>STRD</td>
<td></td>
<td></td>
<td></td>
<td>STRH</td>
<td>STRHT</td>
<td>STRT</td>
</tr>
<tr>
<td>SUB</td>
<td></td>
<td></td>
<td></td>
<td>TBH</td>
<td>TEQ</td>
<td>TST</td>
</tr>
<tr>
<td>UBFX</td>
<td></td>
<td></td>
<td></td>
<td>UXTB</td>
<td>UXTH</td>
<td></td>
</tr>
<tr>
<td>WFE</td>
<td></td>
<td></td>
<td></td>
<td>UMLAL</td>
<td>UMULL</td>
<td>USAT</td>
</tr>
<tr>
<td></td>
<td>WFI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YIELD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CORTEX-M0**

<table>
<thead>
<tr>
<th>BKPT</th>
<th>BLX</th>
<th>ADC</th>
<th>ADD</th>
<th>ADR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>CPS</td>
<td>AND</td>
<td>ASR</td>
<td>B</td>
</tr>
<tr>
<td>DMB</td>
<td></td>
<td>BL</td>
<td>CMP</td>
<td>EOR</td>
</tr>
<tr>
<td>DSB</td>
<td></td>
<td>CMN</td>
<td>CMP</td>
<td>LDR</td>
</tr>
<tr>
<td>ISB</td>
<td></td>
<td>LDRB</td>
<td>LDM</td>
<td>LDRH</td>
</tr>
<tr>
<td>MRS</td>
<td></td>
<td>LDRH</td>
<td>LDRSB</td>
<td>LDRSH</td>
</tr>
<tr>
<td>MSR</td>
<td></td>
<td>LSL</td>
<td>LSR</td>
<td>MOV</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td>MUL</td>
<td>MVN</td>
<td>ORR</td>
</tr>
<tr>
<td>REV</td>
<td></td>
<td>POP</td>
<td>PUSH</td>
<td>ROR</td>
</tr>
<tr>
<td>REV16</td>
<td></td>
<td>STR</td>
<td>STRB</td>
<td>STRH</td>
</tr>
<tr>
<td>SEV</td>
<td></td>
<td>RSB</td>
<td>SBC</td>
<td>STM</td>
</tr>
<tr>
<td>SXTH</td>
<td>UXTB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UXTH</td>
<td>WFE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFI</td>
<td>YIELD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CORTEX-M3**

Present in ARM7TDMI
Code Size
Code size of 32 bits versus 16/8bit MCU’s

- **The instruction size of 8 bit MCU’s is not 8 bits**
  - 8051 is 8 to 24 bits
  - PIC18 is 18 bits
  - PIC16 is 16 bits

- **The instruction size of 16 bit MCU’s is not 16 bits**
  - MSP430 can be up to 32 bits and the extended version can be up to 64 bits
  - PIC24 is 24 bits

- **The instruction size for M0 is mostly 16 bits**
Code size of 32 bits versus 16/8bit MCU’s
### 16-bit multiply example

- **Consider an device with a 10-bit ADC**
  - Basic filtering of data requires a 16-bit multiply operation
  - 16-bit multiply operation is compared below

<table>
<thead>
<tr>
<th>8-bit example</th>
<th>16-bit example</th>
<th>ARM Cortex-M0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, XL ; 2 bytes</td>
<td>MUL AB; 1 byte</td>
<td>MOV R1, &amp;MulOp1</td>
</tr>
<tr>
<td>MOV B, YL ; 3 bytes</td>
<td>ADD A, R1; 1 byte</td>
<td>MOV R2, &amp;MulOp2</td>
</tr>
<tr>
<td>MUL AB; 1 byte</td>
<td>MOV R1, A; 1 byte</td>
<td>MOV SumLo, R3</td>
</tr>
<tr>
<td>MOV R0, A; 1 byte</td>
<td>MOV A, B; 2 bytes</td>
<td>MOV SumHi, R4</td>
</tr>
<tr>
<td>MOV R1, B; 3 bytes</td>
<td>ADDC A, R2; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV A, XL; 2 bytes</td>
<td>MOV R2, A; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV B, YH; 3 bytes</td>
<td>MOV A, XH; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>MUL AB; 1 byte</td>
<td>MOV B, YH; 3 bytes</td>
<td></td>
</tr>
<tr>
<td>ADD A, R1; 1 byte</td>
<td>MUL AB; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV R1, A; 1 byte</td>
<td>MOV A, B; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>MOV A, B; 2 bytes</td>
<td>ADD A, R2; 1 byte</td>
<td></td>
</tr>
<tr>
<td>ADDC A, #0; 2 bytes</td>
<td>MOV R2, A; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV R2, A; 1 byte</td>
<td>MOV A, B; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>MOV A, XH; 2 bytes</td>
<td>ADDC A, #0; 2 bytes</td>
<td></td>
</tr>
<tr>
<td>MOV B, YL; 3 bytes</td>
<td>MOV R3, A; 1 byte</td>
<td></td>
</tr>
<tr>
<td>MOV B, YL; 3 bytes</td>
<td>MOV R1, A; 1 byte</td>
<td></td>
</tr>
</tbody>
</table>

- **Time:** 48 clock cycles*
- **Code size:** 48 bytes

- **Time:** 8 clock cycles
- **Code size:** 8 bytes

- **Time:** 1 clock cycle
- **Code size:** 2 bytes

* 8051 need at least one cycle per instruction byte fetch as they only have an 8-bit interface
What about Data?

- 8 bit microcontrollers do not just process 8 bit data
  - Integers are 16 bits
  - 8 bit microcontroller needs multiple instructions integers
  - C libraries are inefficient
  - Stack size increases
  - Interrupt latency is affected

- Pointers take multiple Bytes.

- M0 can handle Integers in one instruction

- M0 can efficiently process 8 and 16 bit data
  - Supports byte lanes
  - Instructions support half words and bytes.
    LDR, LDRH, LDRB

- M0 has efficient Library support
  - Optimized for M0
What about Data?

- For 16 bit processors have issues with
  - Long integers
  - Floating point types
  - Data transfers between processor registers and memory

- 16 bit processors have 16 bit registers
  - Two registers required for 32 bit transfers
  - Increased stack requirements

- M0 has 32 bit registers and 32 bit memories
  - Less cycles for long integers
  - Good floating point performance
  - Less cycles for data transfers
What addressing modes?

- 16/8 bit processors are limited to 64K of space
  - Data memory limited and segmented
  - Requires banking or extensions to instruction set
  - Memory pointers are extended
    - Require multiple instructions and registers
- All cause increased code space
- M0 has a linear 1G address space
  - 32-bit pointers
  - unsigned or signed 32-bit integers
  - unsigned 16-bit or 8-bit integers
  - signed 16-bit or 8-bit integers
  - unsigned or signed 64-bit integers held in two registers.
Code size increase due to paging

![Graph showing code size increase due to paging](image-url)
Code size increase for large memory model

(Extended program counter and Registers)
Code Size Performance

- a2time
- aiifir
- aiiff
- bitmp
- canrd
- iift
- pnrch
- puwmd
- rspeed

HC08
M0 using microlib
Code Size Performance

- M0 code size is on average 10% smaller than best MSP430 average

![Code size for basic functions graph](image-url)

- Code Size (Bytes)
- Functions: Math8bit, Math16bit, Math32bit, Matrix2dim8bit, Matrix2dim16, Matrixmult, Switch8bit, Switch16bit
- Devices: MSP430, MSP430F5438, MSP430F5438 Large model, Cortex M0
Code Size Performance

- M0 code size is 42% and 36% smaller than best MSP430 generic

![Floating Point and Fir Filter Code Size](chart.png)

- **M0 code size** is smaller than both generic and specific MSP430 models.
Code Size Performance

- M0 code size is 30% smaller than MSP430F5438
What is CoreMark?

- Simple, yet sophisticated
  - Easily ported in hours, if not minutes
  - Comprehensive documentation and run rules

- Free, but not cheap
  - Open C code source download from EEMBC website
  - Robust CPU core functionality coverage

- Dhrystone terminator
  - The benefits of Dhrystone without all the shortcomings
    - Free, small, easily portable
    - CoreMark does real work
CoreMark Workload Features

- Matrix manipulation allows the use of MAC and common math ops
- Linked list manipulation exercises the common use of pointers
- State machine operation represents data dependent branches
- Cyclic Redundancy Check (CRC) is very common embedded function

Testing for:
- A processor’s basic pipeline structure
- Basic read/write operations
- Integer operations
- Control operations
Code Size Performance (CoreMark)

- M0 code size is 16% smaller than generic MSP430
Code Size Performance (CoreMark)

- M0 code size is 53% smaller than PIC24
Code Size Performance (CoreMark)

- M0 code size is 51% smaller than PIC18
Code Size Performance (CoreMark)

- M0 code size is 49% smaller than Atmel AVR8
Code Size Performance (CoreMark)

- M0 code size is 44% smaller than Renesas H8
## Peripheral code

<table>
<thead>
<tr>
<th>Part</th>
<th>Init Code (Bytes)</th>
<th>Data rx code (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVR8 ATmega644</td>
<td>28</td>
<td>32</td>
</tr>
<tr>
<td>MSP430</td>
<td>50</td>
<td>28</td>
</tr>
<tr>
<td>M0 LPC11xx</td>
<td>68</td>
<td>30</td>
</tr>
</tbody>
</table>
Speed Optimization effects

![Graph showing Speed Optimization effects with CoreMark Score and Code Size comparison across t0, t1, t2, and t3 times.](image)
Size Optimization effects

CoreMark Score vs Code Size for different sizes (s0, s1, s2, s3).

Y-axis: Code Size (0-12000)
X-axis: CoreMark Score (1.00 to 1.30)

Legend:
- CoreMark Score
- Code Size
Size Optimization effects

![Size Optimization effects chart](chart.png)
What About Libraries

- 33% reduction using optimized Libs

<table>
<thead>
<tr>
<th></th>
<th>NXP M0 MicroLib</th>
<th>NXP M0 Standard Lib</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Auto BM</strong></td>
<td>Compile</td>
<td>Lib</td>
</tr>
<tr>
<td>a2time</td>
<td>4032</td>
<td>4552</td>
</tr>
<tr>
<td>aifft</td>
<td>4636</td>
<td>6712</td>
</tr>
<tr>
<td>aifirf</td>
<td>3300</td>
<td>4500</td>
</tr>
<tr>
<td>aifft</td>
<td>4348</td>
<td>6636</td>
</tr>
<tr>
<td>basefp</td>
<td>3348</td>
<td>4668</td>
</tr>
<tr>
<td>bitmnp</td>
<td>4776</td>
<td>4412</td>
</tr>
<tr>
<td>canrdr</td>
<td>3272</td>
<td>4412</td>
</tr>
<tr>
<td>idctm</td>
<td>4564</td>
<td>6884</td>
</tr>
<tr>
<td>iirfft</td>
<td>4552</td>
<td>4540</td>
</tr>
<tr>
<td>matrix</td>
<td>6632</td>
<td>4872</td>
</tr>
<tr>
<td>pntrch</td>
<td>3204</td>
<td>4512</td>
</tr>
<tr>
<td>puwmod</td>
<td>3436</td>
<td>4500</td>
</tr>
<tr>
<td>rspeed</td>
<td>2728</td>
<td>4540</td>
</tr>
<tr>
<td>tblook</td>
<td>3612</td>
<td>4864</td>
</tr>
<tr>
<td>ttsprk</td>
<td>5060</td>
<td>4540</td>
</tr>
<tr>
<td><strong>average (8)</strong></td>
<td>3663</td>
<td>4496</td>
</tr>
</tbody>
</table>
Computation Performance

![Computation Performance Chart]

- ARM Cortex-M0
- MSP430FG4619
- PIC18F242
- 8051
Computation Performance

16 bit FIIR filter performance at 1MHz
Computation Performance

CoreMark Score

<table>
<thead>
<tr>
<th>Device</th>
<th>CoreMark (Mark/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC18</td>
<td>0.2</td>
</tr>
<tr>
<td>Renesas (8 bit)</td>
<td>0.4</td>
</tr>
<tr>
<td>AVR8 ATMega644</td>
<td>1.2</td>
</tr>
<tr>
<td>MSP430</td>
<td>1.6</td>
</tr>
<tr>
<td>M0</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Cost
Does the core size matter?

- The M0 core is the smallest cortex core
- About 1/3 of the M3 for similar configuration
- Similar size to 8 bit cores
Core Size Matters

Normalized Cost As a Function of Flash Memory Size
Tools
MCU Tool Solutions

- LPC PRESSO
  - NXP’s Low cost Development Tool Chain

- mbed
  - Rapid Prototyping Online Tool

- IAR Systems, hitex
  - Traditional Feature Rich Tools (third party)
NXP’s FIRST Low Cost Toolchain

LPCXpresso

Eclipse-based IDE

LPCXpresso Starter Board

Evaluation

Product Development
LPCXpresso

- LPCXpresso will provide end-to-end solution from evaluation all the way to product development
- Attractive upgrade options to full blown suites and development boards
- *LPCXpresso will change the perception about NXP’s solution for tools*
- Key competition:
  - Microchip MPLAB
  - Atmel AVR Studio

“LPCXpresso will change the Tool Landscape for NXP”
LPCXpresso Components

- NXP has created the first single perspective Eclipse IDE
- This offers the power and flexibility of Eclipse in combination with a simple and easy to learn user interface
- Supports all NXP products (currently up to 128k)
- LPC3154 HS USB download and debug engine
- LPC134x Target board
Evaluation

- The target board is very simple with one LED and a layout option for USB.
- Traces between the two boards can be cut, to allow SWD connection to any customer target. (Eval target can be reconnected by jumpers.)
Exploration

- Customers can upgrade to full version of Red Suite (Discount coupon)
- Customers can buy an add-on EA base board that connects a wide range of resources to the I/O and peripherals of the LPC13xx. Customers can also upgrade to other EA boards (Discount coupon)
Development

- Traces can be cut and the LPC13xx target board will out of the picture.
- Customers can then use the JTAG connection to download code into their own application board using the same existing IDE and JTAG connector.
- Note: Customers can directly jump to this stage and use LPCXpresso for their complete application development without ever having to upgrade.
mbed LPC1768 Value Proposition

- New users start creating applications in 60 seconds

- Rapid Prototyping with LPC1700 series MCUs
  - Immediate connectivity to peripherals and modules for prototyping LPC1700-based system designs
  - Providing developers with the freedom to be more innovative & productive

- mbed C/C++ Libraries provide API-driven approach to coding
  - High-level interfaces to peripherals enables rock-solid, compact code
  - Built on Cortex Microcontroller Software Interface Standard (CMSIS)

- Download compiled binary by saving to the mbed hardware
  - Just like saving to a USB Flash Drive

- Tools are online - there is nothing to configure, install or update, and everything works on Windows, Mac or Linux

- Hardware in a 40-pin 0.1" pitch DIP form-factor
  - Ideal for solderless breadboard, stripboard and through-hole PCBs
First Experience – Hassle-Free Evaluation

Remove board from the box

Up pops a USB Disk linking to website

Plug it in…

No Installation!

Save to the board and you’re up and running

“Hello World!” in 60 seconds

Compile a program online
mbed Technology

USB Drag ‘n’ Drop Programming Interface
► Nothing to Install: Program by saving binaries
► Works on Windows, Linux, Mac, without drivers
► Links through to mbed.org website

Online Compiler
► Nothing to Install: Browser-based IDE
► Best in class RealView Compiler in the back end
► No code size or production limitations

High-level Peripheral Abstraction Libraries
► Instantly understandable APIs
► Object-oriented hardware/software abstraction
► Enables experimentation without knowing MCU details
Example Beta Projects - Videos

- Rocket Launch
  - http://www.youtube.com/watch?v=zyY451Rb-50&feature=PlayList&p=000FD2855BEA7E90&index=11

- Billy Bass
  - http://www.youtube.com/watch?v=Y6kECR7T4LY

- Voltmeter
  - http://www.youtube.com/watch?v=y_7WxhdLLVU&feature=PlayList&p=000FD2855BEA7E90&index=8

- Knight Rider
  - http://www.youtube.com/watch?v=tmfkLJY-1hc&feature=PlayList&p=000FD2855BEA7E90&index=4

- Bluetooth Big Trak
  - http://www.youtube.com/watch?v=RhC9AbJ_bu8&feature=PlayList&p=000FD2855BEA7E90&index=3

- Scratch Pong
  - http://www.youtube.com/watch?v=aUtYRguMX9g&feature=PlayList&p=000FD2855BEA7E90&index=5
More information

- Available from NXP Distributors and eTools
- Boards cost $99

Learn More:

- [http://www.standardics.nxp.com/support/development.hardware/mbed.lpc176x/](http://www.standardics.nxp.com/support/development.hardware/mbed.lpc176x/)
- [http://MBED.org](http://MBED.org)

Featured Articles:
- [Circuit Cellar](http://Circuit Cellar)
- [Elektor](http://Elektor)
Rapid Prototyping for Microcontrollers
What’s happening in Microcontrollers?

- Microcontrollers are getting **cheap**
  - 32-bit ARM Cortex-M3 Microcontrollers @ $1

- Microcontrollers are getting **powerful**
  - Lots of processing, memory, I/O in one package

- Microcontrollers are getting **interactive**
  - Internet connectivity, new sensors and actuators

- Creates new opportunities for microcontrollers
Rapid Prototyping

- Rapid Prototyping helps industries create new products
  - Control, communication and interaction increasingly define products
  - Development cycles for microelectronics have not kept pace

3D Moulding  3D Printing  2D/3D Design  Web Frameworks
mbed

- Getting Started and Rapid Prototyping with ARM MCUs
  - Complete Targeted Hardware, Software and Web 2.0 Platform

- Dedicated Developer Web Platform
- Lightweight Online Compiler
- Rapid Prototyping for Microcontrollers
- LPC Cortex-M MCU in a Prototyping Form-Factor
mbed Audience

mbed’s focus on Rapid Prototyping has a broad appeal

- Designers new to embedded applications
  - Enables new designs where electronics is not the focus

- Experienced embedded engineers
  - Enables fast proof-of-concepts to reduce risk and push boundaries

- Marketing, distributors and application engineers
  - A consistent platform enables effective and efficient demonstration, support and evaluation of MCUs
Conclusion

- LPC1100 Family Based on the Cortex-M0 core
  - There are many users of 8 and 16 bit microcontrollers that are reluctant to use 32 bit architectures citing either overkill or complexity.
  - The M0 is an architecture that makes this argument irrelevant.
  - The LPC ARM Cortex-M0 family provides a microcontroller that is very low power, has better real-time performance than microcontrollers of lower bit width and provides a bridge to the full spectrum of the LPC families.