

# [www.mientayvn.com](http://www.mientayvn.com)

Khi đọc qua tài liệu này, nếu phát hiện sai sót hoặc nội dung kém chất lượng xin hãy thông báo để chúng tôi sửa chữa hoặc thay thế bằng một tài liệu cùng chủ đề của tác giả khác.

Bạn có thể tham khảo nguồn tài liệu được dịch từ tiếng Anh tại đây:

[http://mientayvn.com/Tai\\_lieu\\_da\\_dich.html](http://mientayvn.com/Tai_lieu_da_dich.html)

Thông tin liên hệ:

Yahoo mail: [thanhlam1910\\_2006@yahoo.com](mailto:thanhlam1910_2006@yahoo.com)

Gmail: [frbwirthes@gmail.com](mailto:frbwirthes@gmail.com)

**Theo yêu cầu của khách hàng, trong một năm qua, chúng tôi đã dịch qua 16 môn học, 34 cuốn sách, 43 bài báo, 5 sổ tay (chưa tính các tài liệu từ năm 2010 trở về trước) Xem ở đây**

**DỊCH VỤ  
DỊCH  
TIẾNG  
ANH  
CHUYÊN  
NGÀNH  
NHANH  
NHẤT VÀ  
CHÍNH  
XÁC  
NHẤT**

Chỉ sau một lần liên lạc, việc dịch được tiến hành

Giá cả: có thể giảm đến 10 nghìn/1 trang

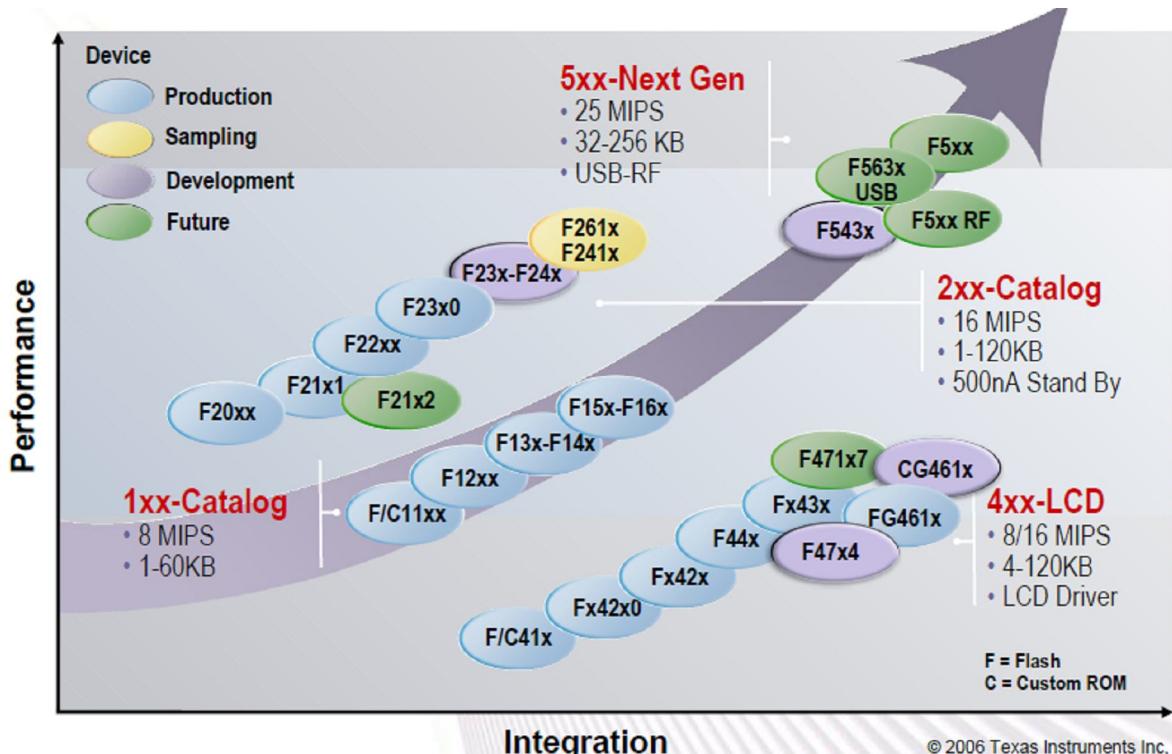
Chất lượng:Tạo dựng niềm tin cho khách hàng bằng công nghệ 1.Bạn thấy được toàn bộ bản dịch; 2.Bạn đánh giá chất lượng. 3.Bạn quyết định thanh toán.

## CHƯƠNG II: GIỚI THIỆU HỘ VI ĐIỀU KHIỂN MSP430



### 2.1 Giới thiệu

MSP430 là một sự kết hợp chặt chẽ của một CPU RISC 16 bit, những khối ngoại vi, và hệ thống xung linh hoạt. MSP430 đã đưa ra được những giải pháp tốt cho những nhu cầu ứng dụng với nhiều phiên bản khác nhau. MSP430 có một số phiên bản như: MSP430x1xx, MSP430x2xx, MSP430x3xx, MSP430x4xx, MSP430x5xx.

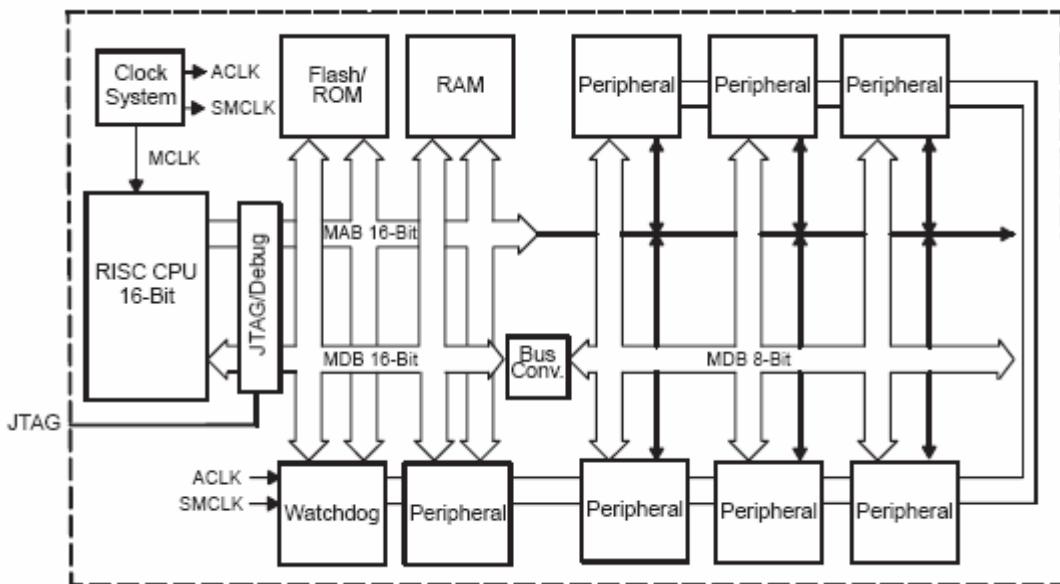


Hình 2.1: Một số phiên bản hộ vi điều khiển MSP430

Dưới đây là những đặc điểm tổng quát của hộ vi điều khiển MSP430 (tôi chỉ trình bày một số đặc điểm cơ bản):

- ✚ Kiến trúc nguồn điện cực thấp để mở rộng tuổi thọ của Pin
  - $1\mu\text{A}$  duy trì RAM
  - $0.8 \mu\text{A}$  chế độ xung thời gian thực
  - $250 \mu\text{A} / \text{MIPS}$  tích cực
- ✚ Xử lý tín hiệu tương tự với hiệu xuất cao:
  - 12-bit hoặc 10-bit ADC – 200Ksps, cảm biến nhiệt, V(Ref).
  - 12-bit kép DAC.

- ✚ 16 bit RISC CPU cho phép được nhiều ứng dụng, thể hiện một phần ở kích thước code lập trình.
  - Thanh ghi lớn nên loại trừ được trường hợp tắt nghẽn tập tin khi đang làm việc.
  - Thiết kế nhỏ gọn làm giảm lượng tiêu thụ điện và giảm giá thành.
  - Tối ưu hóa cho những chương trình ngôn ngữ bậc cao như C, C++
  - Có 7 chế độ định địa chỉ.
  - Khả năng ngắt theo vec-to lớn.
- ✚ Trong lập trình cho bộ nhớ Flash cho phép thay đổi Code một cách linh hoạt, phạm vi rộng, bộ nhớ Flash còn có thể lưu lại được nhật ký của dữ liệu.



Hình 2.2: Cấu trúc vi điều khiển MSP430

## 2.2 Không gian địa chỉ

Cấu trúc vi điều khiển MSP430 có một địa chỉ không gian nhớ được chia sẻ với các thanh ghi chức năng đặc biệt (SFRs), các bộ ngoại vi, RAM, và bộ nhớ Flash/ROM được biểu diễn trên hình vẽ. Việc truy cập mã chương trình luôn luôn được thực hiện trên một địa chỉ chẵn. Dữ liệu có thể được truy cập như là những byte hay những từ.

Không gian địa chỉ nhớ có thể mở rộng hơn nữa cho những kế hoạch khác.

		Access
1FFFFh	Flash/ROM Interrupt	Word/Byte
10000h		
0FFFFh	Vector Table	Word/Byte
OFFE0h		
OFFDFh	Flash/ROM	Word/Byte
0200h	RAM	Word/Byte
01FFh	16-Bit Peripheral Modules	Word
0100h		
0FFh	8-Bit Peripheral Modules	Byte
010h		
0Fh	Special Function Registers	Byte
0h		

Hình 2.3: Sơ đồ bộ nhớ

### 2.2.1 Flash/ROM

Địa chỉ bắt đầu của Flash/ROM phụ thuộc vào số lượng Flash/ROM hiện có và thay đổi tùy theo loại chip. Địa chỉ kết thúc cho Flash/ROM là 0FFFFh. Flash có thể được sử dụng cho cả mã và chương trình. Những bảng từ hay byte có thể được cắt và sử dụng trong Flash/ROM mà không cần bảng sao chép tới RAM trước khi sử dụng chúng.

### 2.2.2 RAM

RAM có địa chỉ bắt đầu tại 0200h. Địa chỉ kết thúc của RAM phụ thuộc vào số lượng RAM có và thay đổi tùy thuộc vào từng dòng vi điều khiển. RAM có thể được sử dụng cho cả mã và dữ liệu.

### 2.2.3 Những khối ngoại vi

Những module giao tiếp ngoại vi được xếp xắp vào không gian địa chỉ. Không gian địa chỉ từ 0100h tới 01FFh được dành riêng cho module ngoại vi 16 bit. Những module này có thể được truy cập với những từ chỉ dẫn(lệnh).

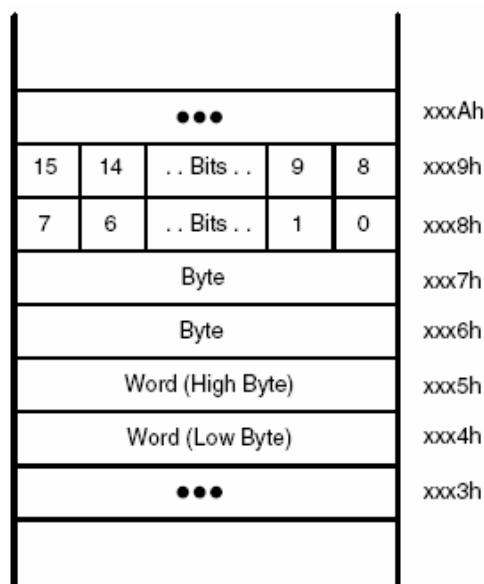
Không gian địa chỉ từ 010h tới 0FFh được dành riêng cho module ngoại vi 8 bit.

#### 2.2.4 Những thanh ghi chức năng đặc biệt (SFR)

Một vài chức năng ngoại vi được cấu hình trong thanh ghi chức năng đặc biệt. Những thanh ghi chức năng đặc biệt được nằm trong 16 byte thấp của không gian địa chỉ. Những SFR phải được truy cập bằng việc sử dụng câu lệnh byte.

#### 2.2.5 Truy cập bộ nhớ

Những byte được nằm tại những địa chỉ chẵn hay lẻ. Những từ chỉ nằm tại địa chỉ chẵn được biểu diễn trong hình 1-3. Khi sử dụng từ chỉ dẫn, chỉ những địa chỉ chẵn có thể được sử dụng. Những byte thấp của một từ luôn luôn là một địa chỉ chẵn. Byte cao ở tại địa chỉ lẻ tiếp theo. Ví dụ, nếu một từ dữ liệu nằm tại địa chỉ xxx4h, kết thúc byte thấp của từ dữ liệu nằm tại địa chỉ xxx4h, và byte cao của từ đó nằm tại địa chỉ xxx5h.



Hình 2.4: Những bit, những byte, và những từ trong một trật tự byte bộ nhớ

## CHƯƠNG II: TẬP LỆNH VI ĐIỀU KHIỂN

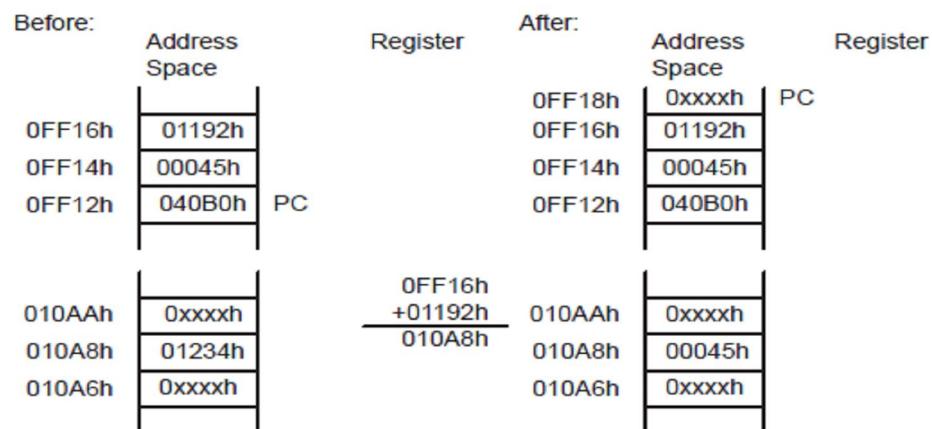
Lecture

### 2.1 Các cách định địa chỉ

#### 2.1.1 Định địa chỉ trực tiếp (Immediate Mode)

MOV #30H, R0 ; đưa giá trị 30h vào thanh ghi R0

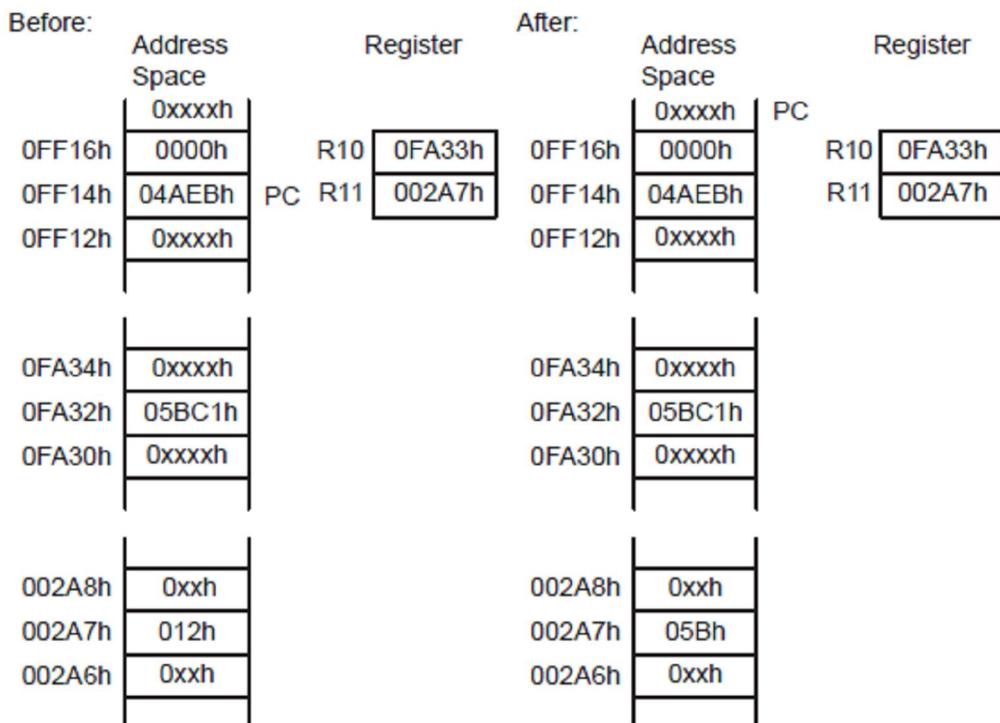
Example: MOV #45h, TON1



#### 2.1.2 Định địa chỉ gián tiếp thanh ghi (Indirect Register Mode)

MOV @R10, 0(R0) ; đưa địa chỉ giá trị nội dung của thanh ghi R10 vào địa chỉ có chứa nội dung thanh ghi R0. Nhưng giá trị thanh ghi không thay đổi.

Example: MOV.B @R10, 0 (R11)

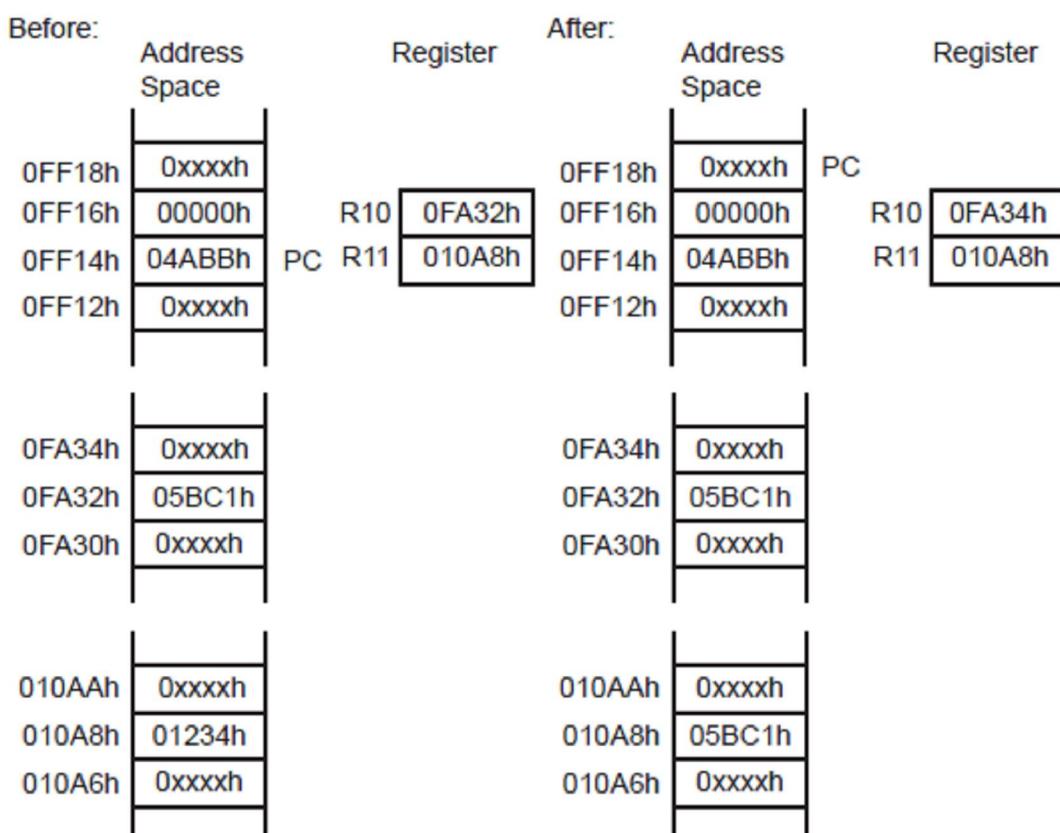


### 2.1.3 Định địa chỉ gián tiếp tự tăng (Indirect Autoincrement Mode)

MOV @R10+,0(R0); lấy nội dung của thanh ghi R10 vào thanh ghi R0 và đồng thời tăng địa chỉ thanh ghi R10 lên 2.

Ví dụ lúc đầu Thanh ghi R10 có Chứa địa chỉ của ô nhớ (123h) có chứa giá trị là 10h, thanh ghi R0 có chứa địa chỉ là 0AFH thì sao khi thực hiện lệnh đó ta có kết quả như sau: địa chỉ con trỏ của R10 tăng lên 125h, còn địa chỉ con trỏ của R0 không đổi là 0AFH, Nhưng nó có chứa giá trị là 10h.

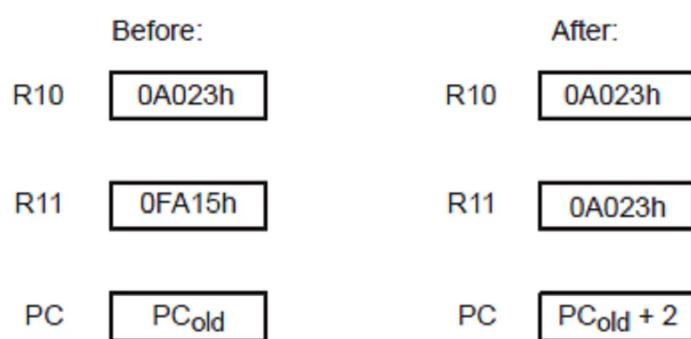
Example: MOV @R10+,0 (R11)



### 2.1.4 Định địa chỉ trực tiếp thanh ghi (Immediate Mode)

MOV R0,R1 ; đưa giá trị thanh ghi R0 vào thanh ghi R1.

Example: MOV R10,R11



### 2.1.5 Định địa chỉ tuyệt đối (Absolute Mode)

MOV &EDE,&TONI ; đưa giá trị của địa chỉ có chứa nhãn EDE vào địa chỉ có chứa nhãn TONI.

Ví dụ EDE có địa chỉ là OFF0h chứa giá trị là 1234h, TONI có địa chỉ là 1FFh có giá trị bất kỳ. Sau khi thực hiện lệnh thì TONI có giá trị là 1234h.

### 2.1.6 Định địa chỉ giữ các biến (Symbolic Mode)

MOV EDE,TONI ; đưa giá trị của biến có địa chỉ của biến EDE vào biến TONI.

Ví dụ EDE có giá trị 10h, TONI có giá trị bất kỳ. Sau khi thực hiện lệnh TONI có giá trị là 10h.

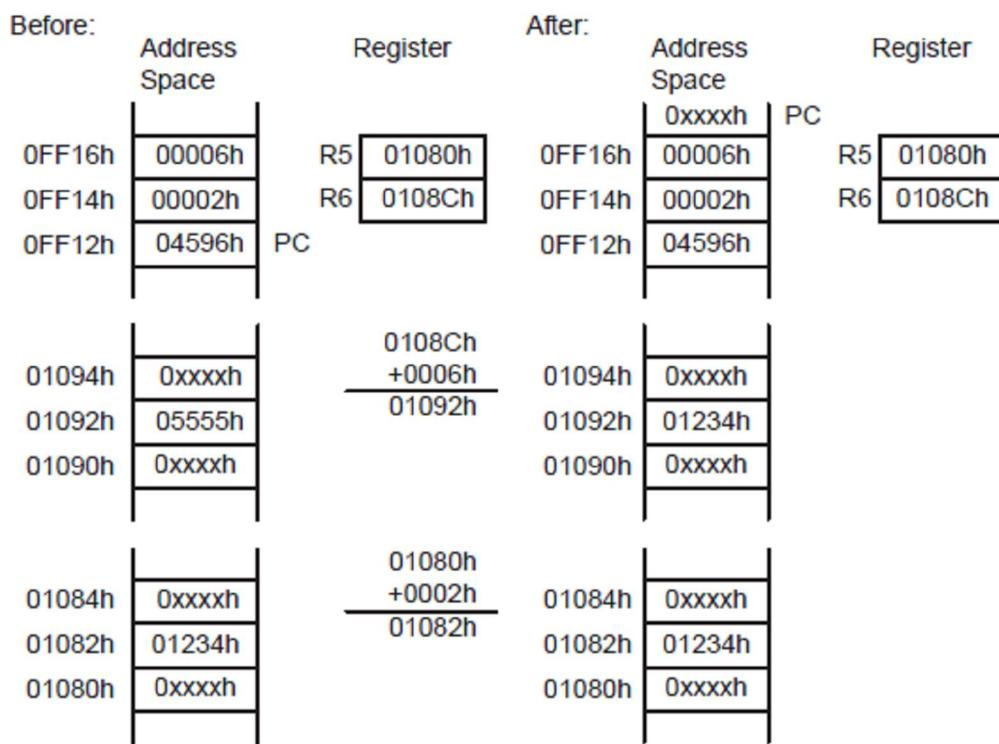
Example:      MOV EDE,TONI ;Source address EDE = 0F016h  
                  ;Dest. address TONI=01114h

Before:	Address Space	Register	After:	Address Space	Register
OFF16h	011FEh		OFF16h	0xxxxh	PC
OFF14h	0F102h		OFF14h	011FEh	
OFF12h	04090h	PC	OFF12h	0F102h	
				04090h	
OF018h	0xxxxh		OF018h	0xxxxh	
OF016h	0A123h		OF016h	0A123h	
OF014h	0xxxxh		OF014h	0xxxxh	
01116h	0xxxxh		01116h	0xxxxh	
01114h	05555h		01114h	0A123h	
01112h	0xxxxh		01112h	0xxxxh	

### 2.1.7 Định địa chỉ con trỏ (Indexed Mode)

MOV 2(R5),3(R6) ; đưa giá trị tại địa chỉ của R5 +2 vào địa chỉ R6+3.

Example: MOV 2 (R5) , 6 (R6) ;



Tóm lại, các cách định địa chỉ rất quan trọng và cần thiết trong quá trình tìm hiểu vi điều khiển và tập lệnh của nó. Để dễ nhớ chúng ta có thể xem bảng tóm tắt các chế độ định địa chỉ ở bên dưới.

Bảng 2.1 Tóm tắt các chế độ định địa chỉ

ADDRESS MODE	S	D	SYNTAX	EXAMPLE	OPERATION
Register	●	●	MOV Rs,Rd	MOV R10,R11	R10 --> R11
Indexed	●	●	MOV X(Rn),Y(Rm)	MOV 2(R5),6(R6)	M(2+R5)--> M(6+R6)
Symbolic (PC relative)	●	●	MOV EDE,TONI		M(EDE) --> M(TONI)
Absolute	●	●	MOV &MEM,&TCDAT		M(MEM) --> M(TCDAT)
Indirect	●		MOV @Rn,Y(Rm)	MOV @R10,Tab(R6)	M(R10) --> M(Tab+R6)
Indirect autoincrement	●		MOV @Rn+,Rm	MOV @R10+,R11	M(R10) --> R11 R10 + 2 --> R10
Immediate	●		MOV #X,TONI	MOV #45,TONI	#45 --> M(TONI)

NOTE: S = source D = destination

## 2.2 Các lệnh thông dụng

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction	
0	0	0	1	0	0	opcode		B/W	As	source		Single-operand arithmetic					
0	0	0	1	0	0	0	0	0	B/W	As	source		<b>RRC</b> Rotate right through carry				
0	0	0	1	0	0	0	0	1	0	As	source		<b>SWPB</b> Swap bytes				
0	0	0	1	0	0	0	0	1	0	B/W	As	source		<b>RRA</b> Rotate right arithmetic			
0	0	0	1	0	0	0	0	1	1	0	As	source		<b>SXT</b> Sign extend byte to word			
0	0	0	1	0	0	1	0	0	B/W	As	source		<b>PUSH</b> Push value onto stack				
0	0	0	1	0	0	1	0	1	0	As	source		<b>CALL</b> Subroutine call; push PC and move source to PC				
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	<b>RETI</b> Return from interrupt; pop SR then pop PC		
0	0	1	condition		10-bit signed offset					Conditional jump; PC = PC + 2×offset							
0	0	1	0	0	0	10-bit signed offset					<b>JNE/JNZ</b> Jump if not equal/zero						
0	0	1	0	0	1	10-bit signed offset					<b>JEQ/JZ</b> Jump if equal/zero						
0	0	1	0	1	0	10-bit signed offset					<b>JNC/JLO</b> Jump if no carry/lower						
0	0	1	0	1	1	10-bit signed offset					<b>JC/JHS</b> Jump if carry/higher or same						
0	0	1	1	0	0	10-bit signed offset					<b>JN</b> Jump if negative						
0	0	1	1	0	1	10-bit signed offset					<b>JGE</b> Jump if greater or equal (N == V)						
0	0	1	1	1	0	10-bit signed offset					<b>JL</b> Jump if less (N != V)						
0	0	1	1	1	1	10-bit signed offset					<b>JMP</b> Jump (unconditionally)						
opcode				source		Ad	B/W	As	destination		Two-operand arithmetic						
0	1	0	0	source		Ad	B/W	As	destination		<b>MOV</b> Move source to destination						
0	1	0	1	source		Ad	B/W	As	destination		<b>ADD</b> Add source to destination						
0	1	1	0	source		Ad	B/W	As	destination		<b>ADDC</b> Add w/carry: dst += (src+C)						
0	1	1	1	source		Ad	B/W	As	destination		<b>SUBC</b> Subtract w/ carry: dst -= (src+C)						
1	0	0	0	source		Ad	B/W	As	destination		<b>SUB</b> Subtract; dst -= src						
1	0	0	1	source		Ad	B/W	As	destination		<b>CMP</b> Compare; (dst-src); discard result						
1	0	1	0	source		Ad	B/W	As	destination		<b>DADD</b> Decimal (BCD) addition: dst += src						
1	0	1	1	source		Ad	B/W	As	destination		<b>BIT</b> Test bits; (dst & src); discard result						
1	1	0	0	source		Ad	B/W	As	destination		<b>BIC</b> Bit clear; dest &= ~src						
1	1	0	1	source		Ad	B/W	As	destination		<b>BIS</b> "Bit set" - logical OR; dst  = src						
1	1	1	0	source		Ad	B/W	As	destination		<b>XOR</b> Bitwise XOR; dst ^= src						
1	1	1	1	source		Ad	B/W	As	destination		<b>AND</b> Bitwise AND; dst &= src						

As	src	Syntax	Description
0	n	<b>Rn</b>	Register direct. The operand is the contents of Rn.
1	n	<b>x(Rn)</b>	Indexed. The operand is in memory at address Rn+x.
2	n	<b>@Rn</b>	Register indirect. The operand is in memory at the address held in Rn.
3	n	<b>@Rn+</b>	Indirect autoincrement. As above, then the register is incremented by 1 or 2.
Addressing modes using R0 (PC)			
3	0	#x	Immediate. @PC+ The operand is the next word in the instruction stream.
Addressing modes using R2 (SP) and R3 , special-case decoding			
1	2	<b>&amp;LABEL</b>	Absolute. The operand is in memory at address x.
2	2	<b>#4</b>	Constant. The operand is the constant 4.
3	2	<b>#8</b>	Constant. The operand is the constant 8.
0	3	<b>#0</b>	Constant. The operand is the constant 0.
1	3	<b>#1</b>	Constant. The operand is the constant 1. There is no index word.
2	3	<b>#2</b>	Constant. The operand is the constant 2.
3	3	<b>#-1</b>	Constant. The operand is the constant -1.

## CHƯƠNG III: TRÌNH BIÊN DỊCH IAR EMBEDDED WORKBENCH



### 3. 1 Sơ lược

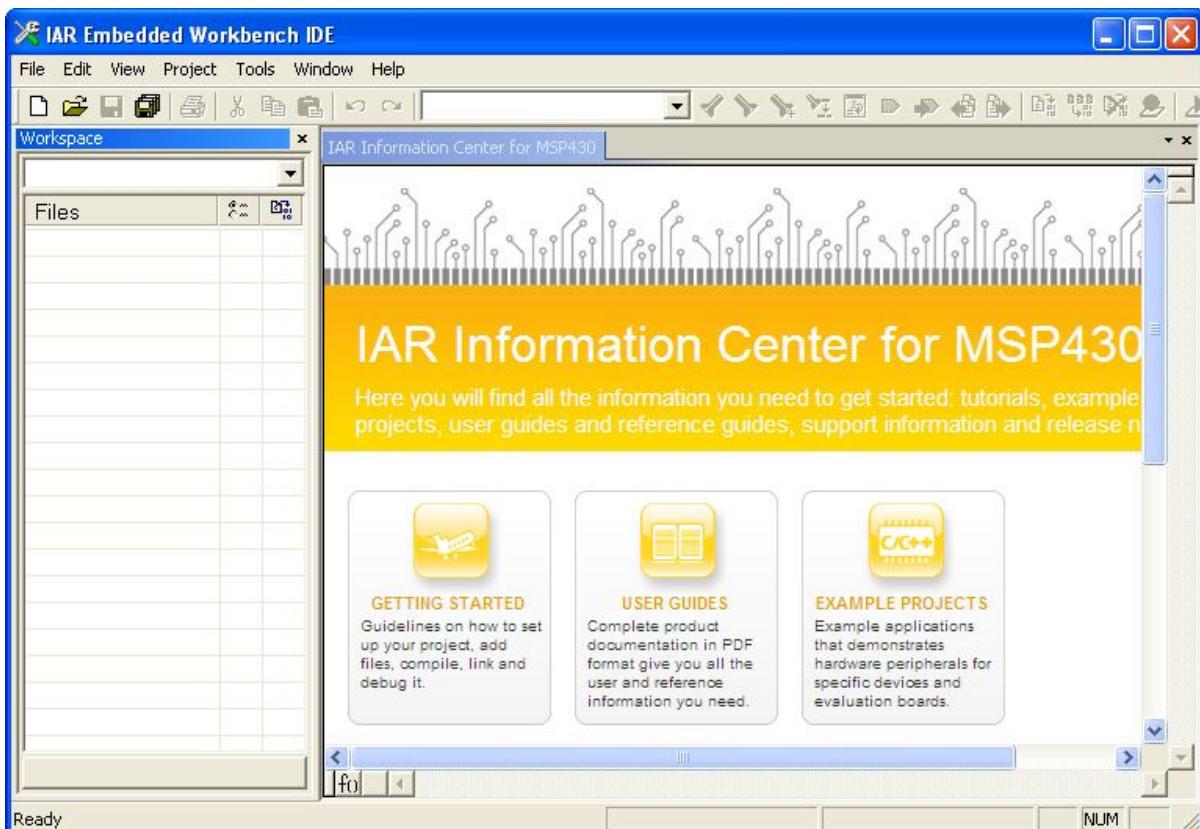
IAR Embedded Workbench là chương trình biên dịch được cung cấp IAR SYSTEMS. Có 3 phiên bản: Kickstart Version – Free, Baseline Version ~ \$795 và Full Version ~ \$2695.

- ✚ Kickstart Version – Free
  - Giới hạn 4KB trong code C.
  - Không giới hạn code asm.
  - Hỗ trợ từ web của hãng TI.
- ✚ Baseline Version ~ \$795
  - Giới hạn 12KB trong code C.
  - Không giới hạn code asm.
  - Hỗ trợ bởi IAR.
- ✚ Full Version ~ \$2695
  - Không giới hạn tất cả code.
  - Hỗ trợ bởi IAR.

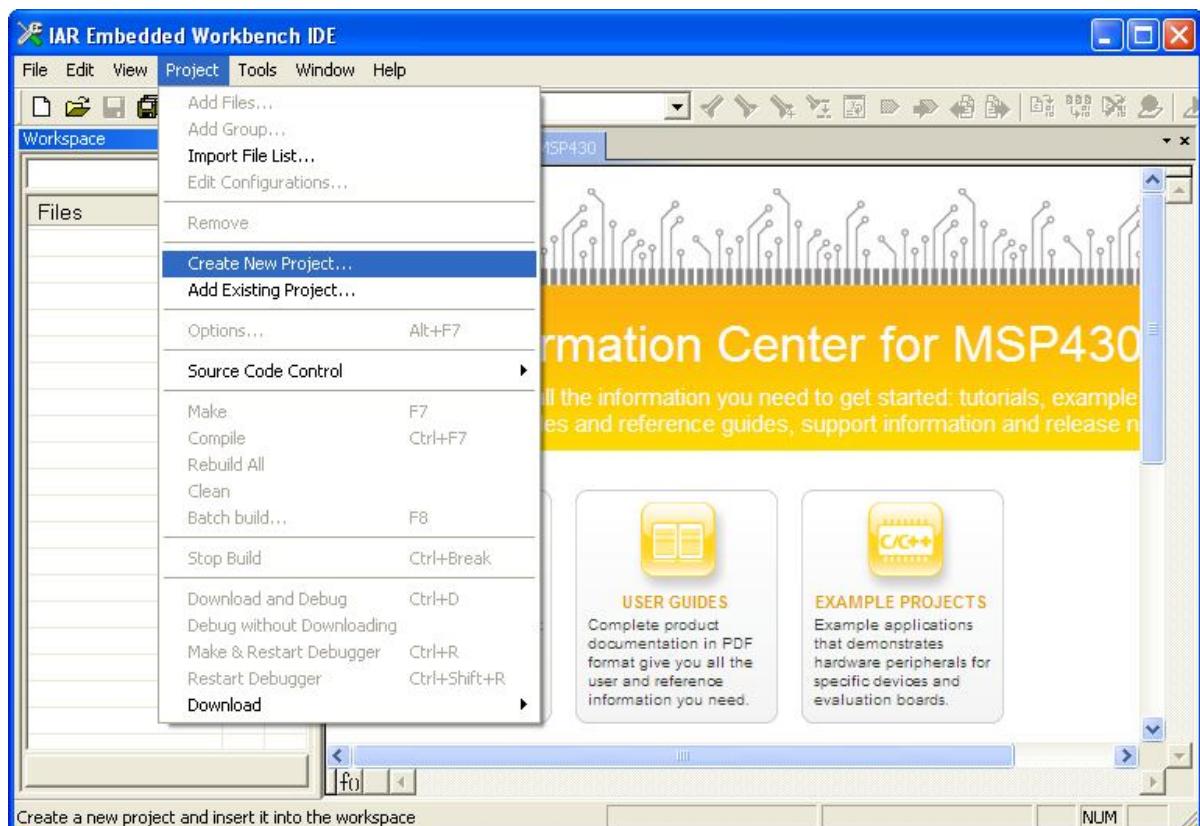


### 3. 2 Cách tạo dự án

Sau khi cài đặt, chúng ta có thể khởi động IAR Embedded Workbench bằng cách vào *Start/All Programs/IAR Systems/IAR Embedded Workbench Kickstart for MSP430 4.21/ IAR Embedded Workbench*. Giao diện chính khi chúng ta khởi động phần mềm lên như sau:



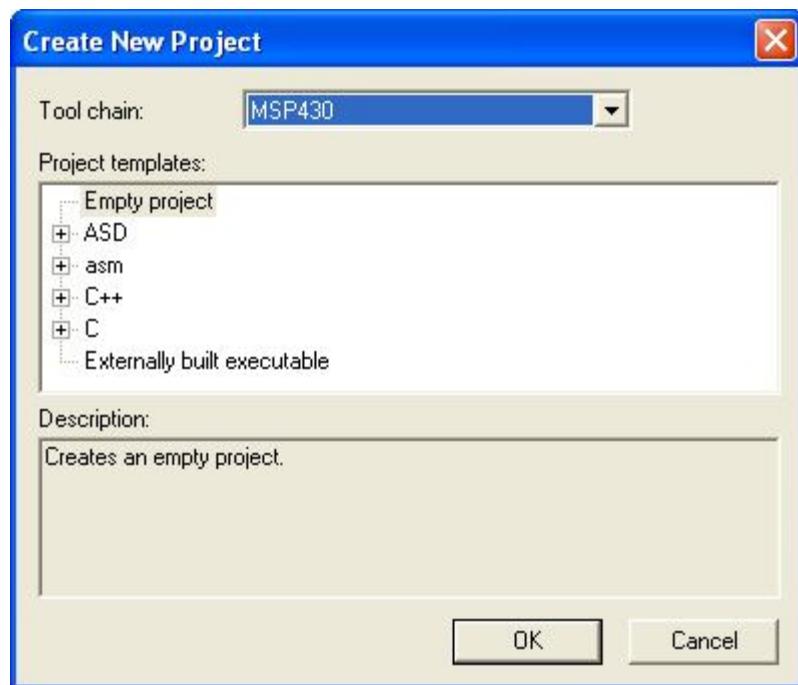
Hình 3-1 Giao diện làm việc.



Hình 3-2 Tạo một dự án.

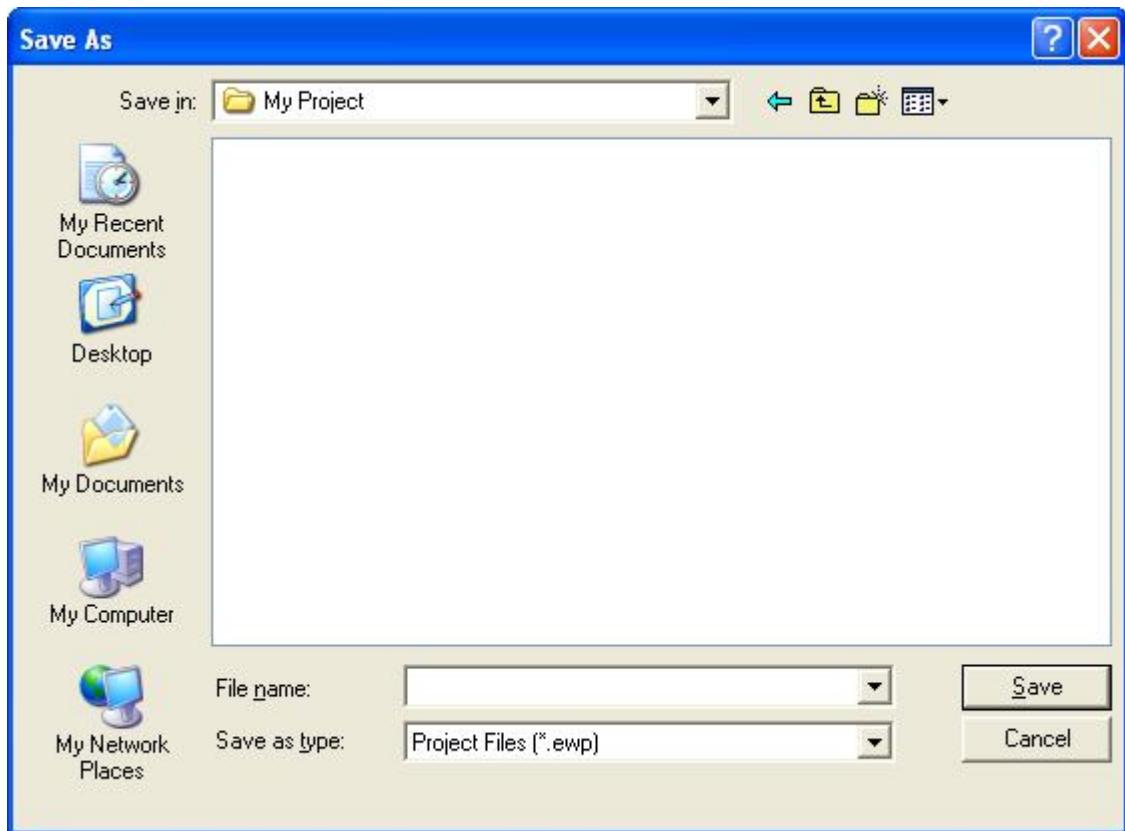
Từ Menu chọn *Project \Create New Project*, một cửa sổ hiện ra ta thực hiện các bước sau:

Chọn ngôn ngữ lập trình trong dự án (ví dụ: asm, c, c++...), xong ta chọn *OK* để tiếp tục.



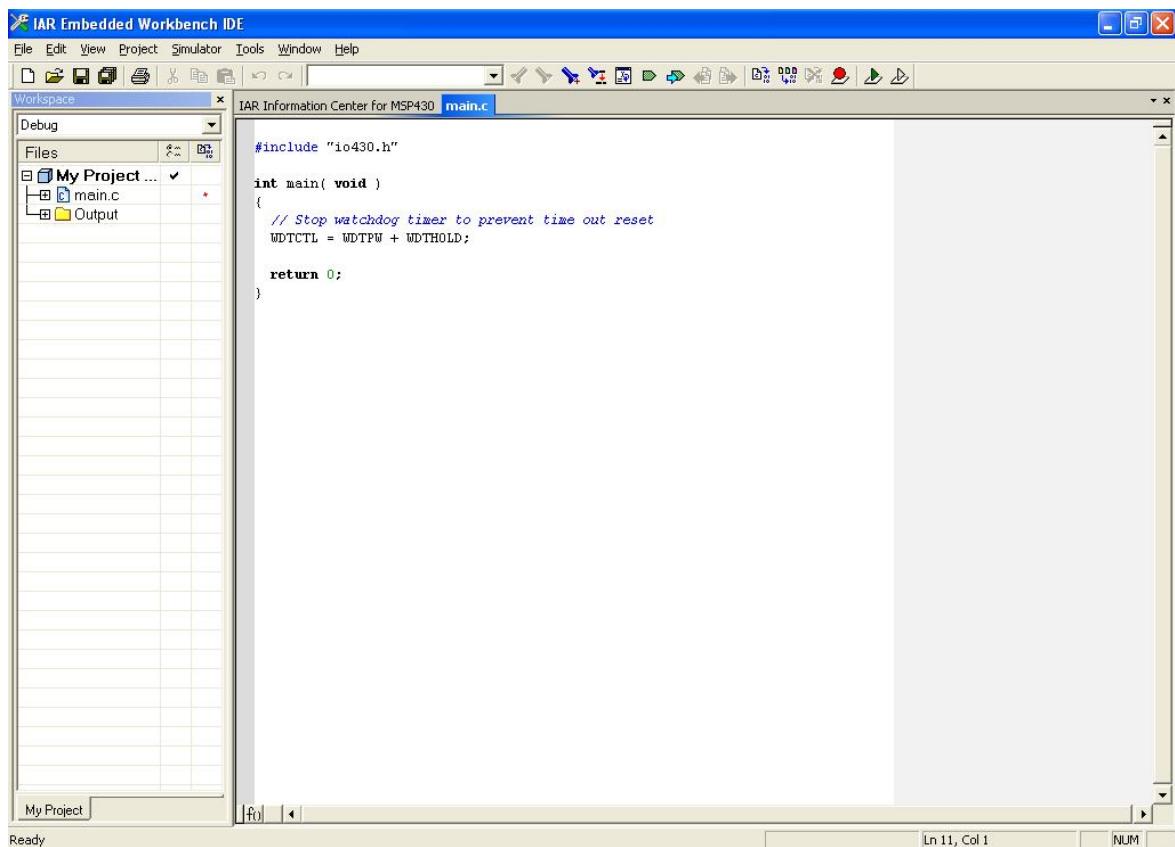
Hình 3-3 Chọn ngôn ngữ để viết dự án.

Sau khi chọn ngôn ngữ để biên dịch. Một cửa sổ *Save As* hiện ra để ta đặt tên cho dự án và đường dẫn lưu trữ. Sau khi thực hiện xong ta *Save* dự án lại.



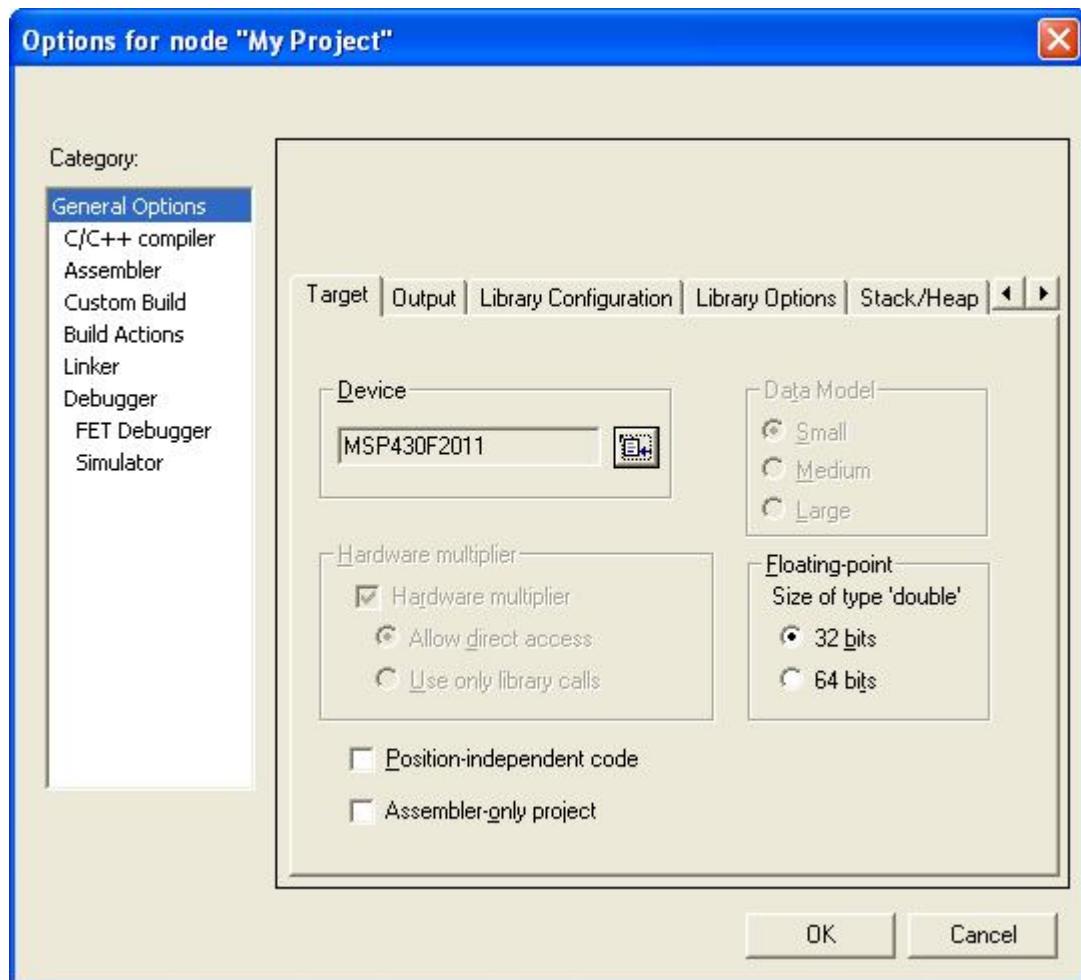
Hình 3-4 Đặt tên và lưu trữ dự án.

Khi ta lưu lại dự án xong chương trình sẽ hiện ra như hình 3-5 khi đó có một file mẫu là: main.c trong dự án vừa tạo ra.



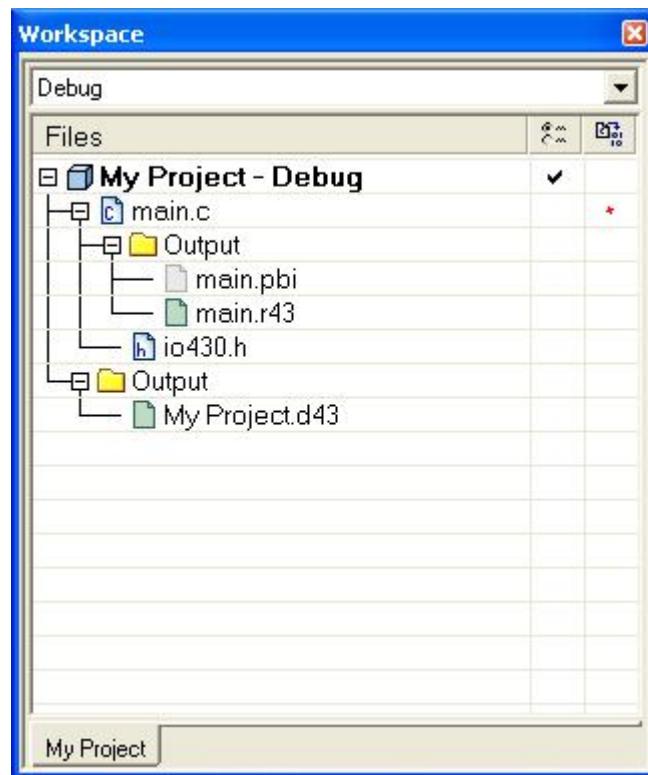
Hình 3-5 Thiết lập dự án đã hoàn thành.

Tiếp theo ta chọn loại vi điều khiển để biên dịch. Nhấp chuột phải vào *My Project - Debug* rồi chọn *Options...*. Ta chọn *General Options* chọn thẻ *Target* chọn *Device* để chọn loại vi điều khiển để biên dịch.



Hình 3-6 Cho loại vi điều khiển trong họ MSP430.

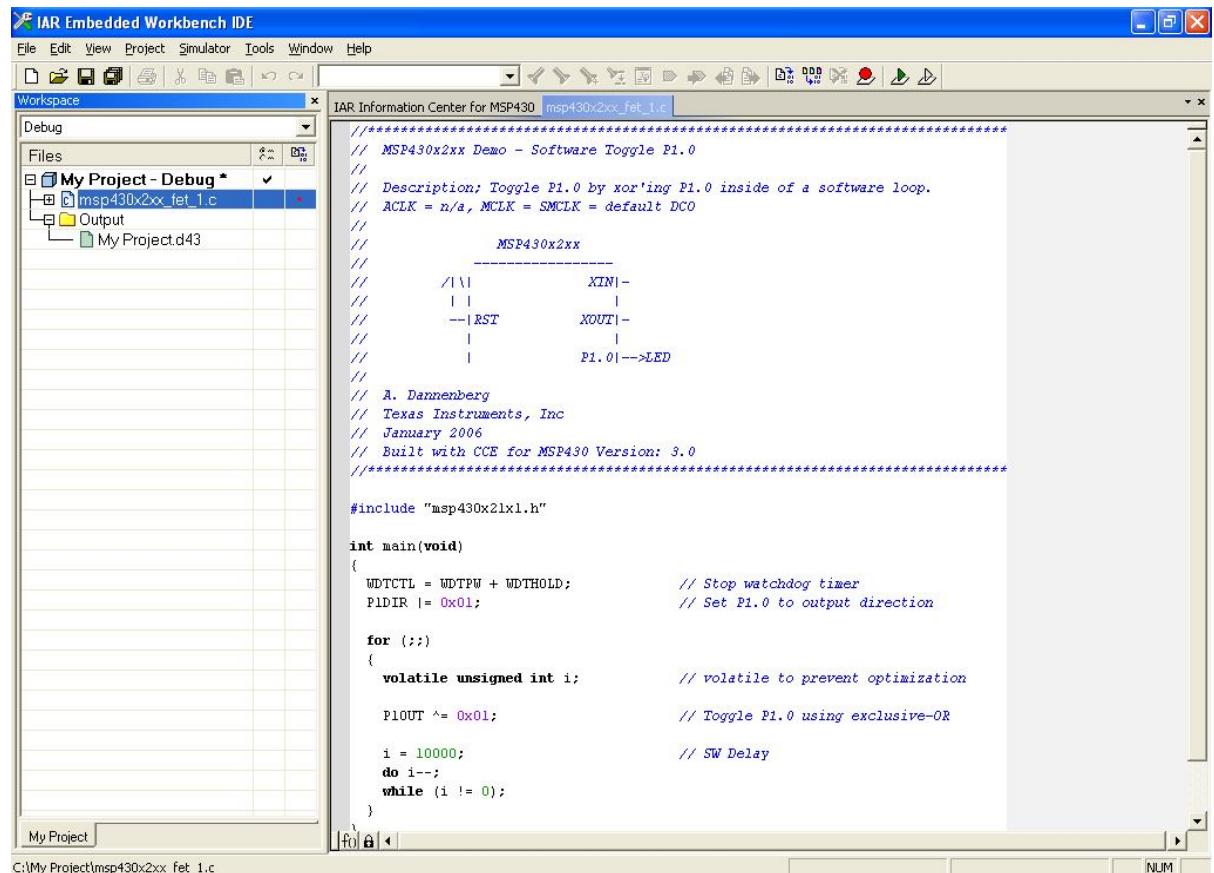
Chọn *File/New/File* để tạo một tập tin mới, rồi tiếp tục chọn *File/Save as* để lưu tập tin (có phần mở rộng \*.asm hoặc \*c...) vào thư mục chứa dự án. Nhấp chuột phải vào *My Project - Debug* rồi chọn *Add/Add files...* để thêm tập tin vào dự án.



Hình 3-7 Cửa sổ quản lý dự án.

Sau khi hoàn thành chúng ta có thể soạn thảo chương trình vào tập tin vừa tạo.

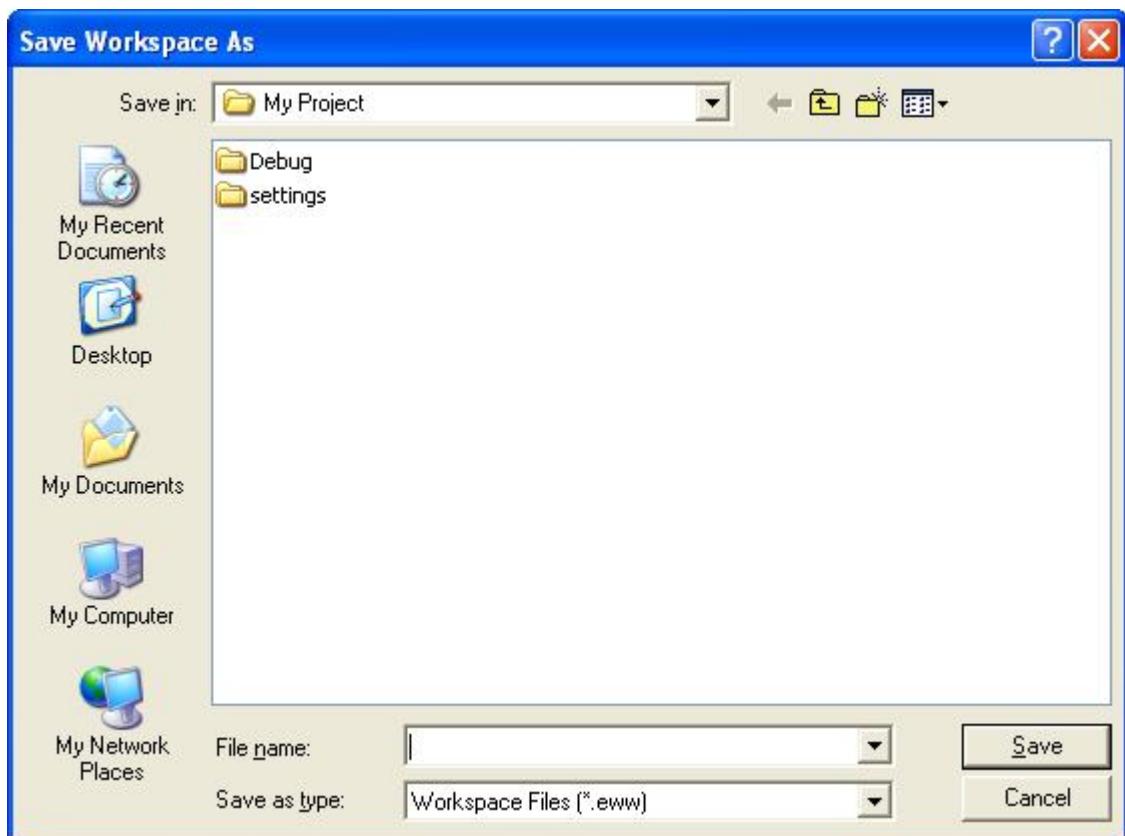
Một dự án hoàn thành như sau:



Hình 3-8 Ví dụ 1 dự án hoàn thành.

### 3.3 Biên dịch và mô phỏng dự án

Sau khi soạn thảo xong chương trình nguồn. Bạn tiến hành biên dịch chương trình. Để biên dịch dự án chúng ta chọn *Project/Make*. Có một cửa sổ bắt buộc chúng ta phải lưu lại Workspace.



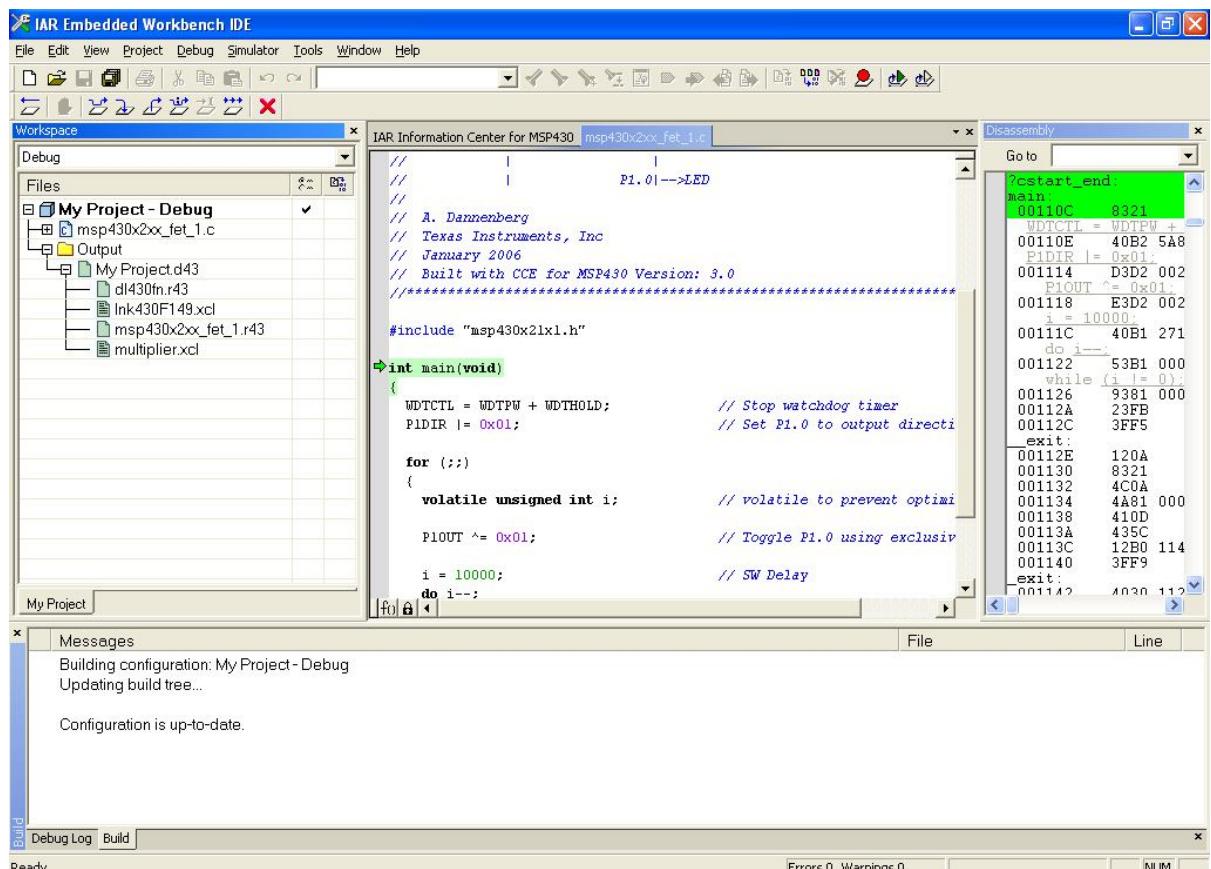
Hình 3-9 Lưu lại Workspace trước khi biên dịch.

Sau khi lưu lại Workspace chúng ta sẽ thấy thông báo hiện ra báo chương trình có lỗi hay không.



Hình 3-10 Cửa sổ biên dịch thành công.

Sau khi biên dịch thành công chúng ta có thể mô phỏng thử dự án. Công việc này khá quan trọng, nó cho phép ta kiểm tra và phát hiện ra các vấn đề về giải thuật của chương trình mà trình biên dịch không làm được. Để tiến hành mô phỏng ta chọn *Project/Download and Debug (Ctrl + D)* để Debug – mô phỏng dự án.



Hình 3-11 Cửa sổ Debug của chương trình.

Ta thấy các công cụ Debug hiện ra như sau:



**Stop Debug:** Thoát khỏi chế độ Debug.



**Go:**



**Next Statement:**



**Step Out:** Chạy toàn chương trình, không thê quan sát biến, thanh ghi khi công cụ này được kích hoạt.



**Step Into:** Chạy từng bước chương trình do người dùng kích hoạt, mỗi nhấp chuột vào biểu tượng này hay ấn phím F7 thì chương trình sẽ thực thi một lệnh.

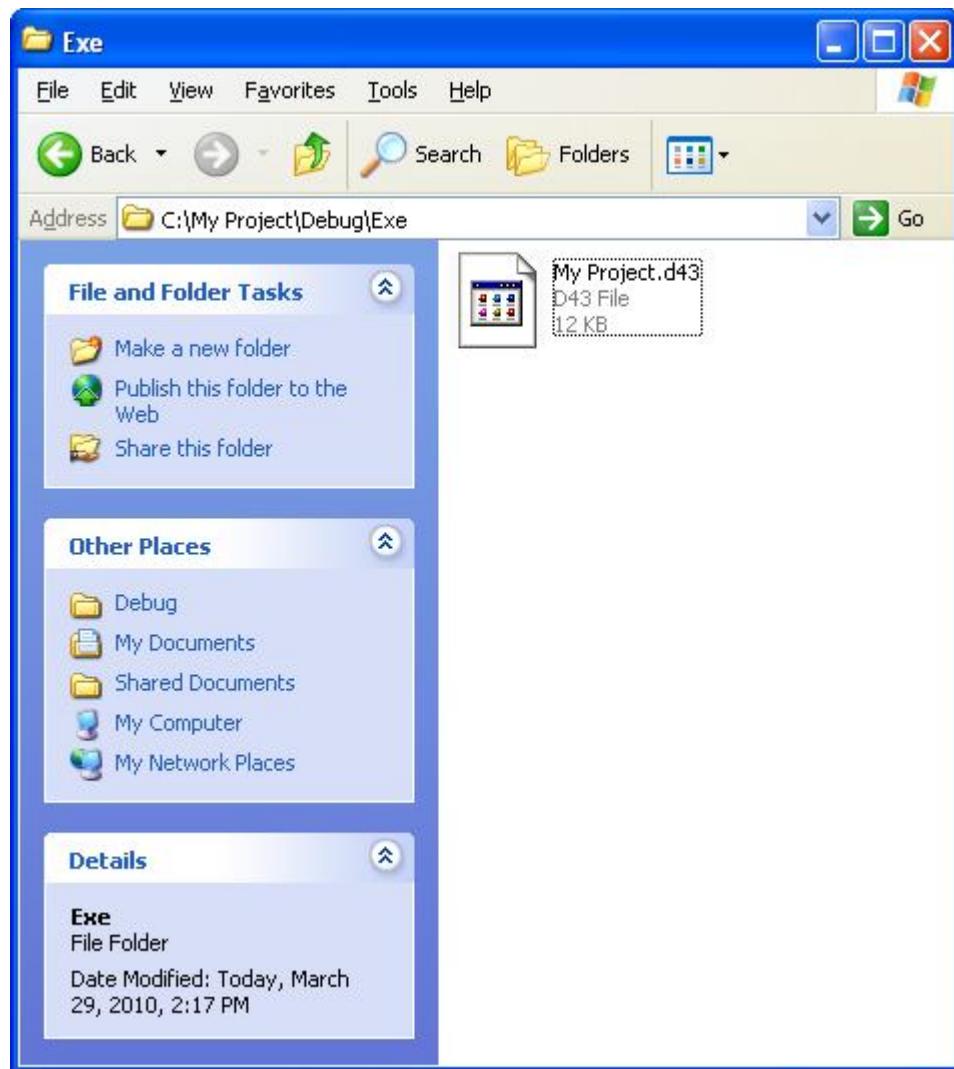


**Step Over:** Tương tự như **Step Into** nhưng nó xem chương trình con hay macro như là một lệnh.



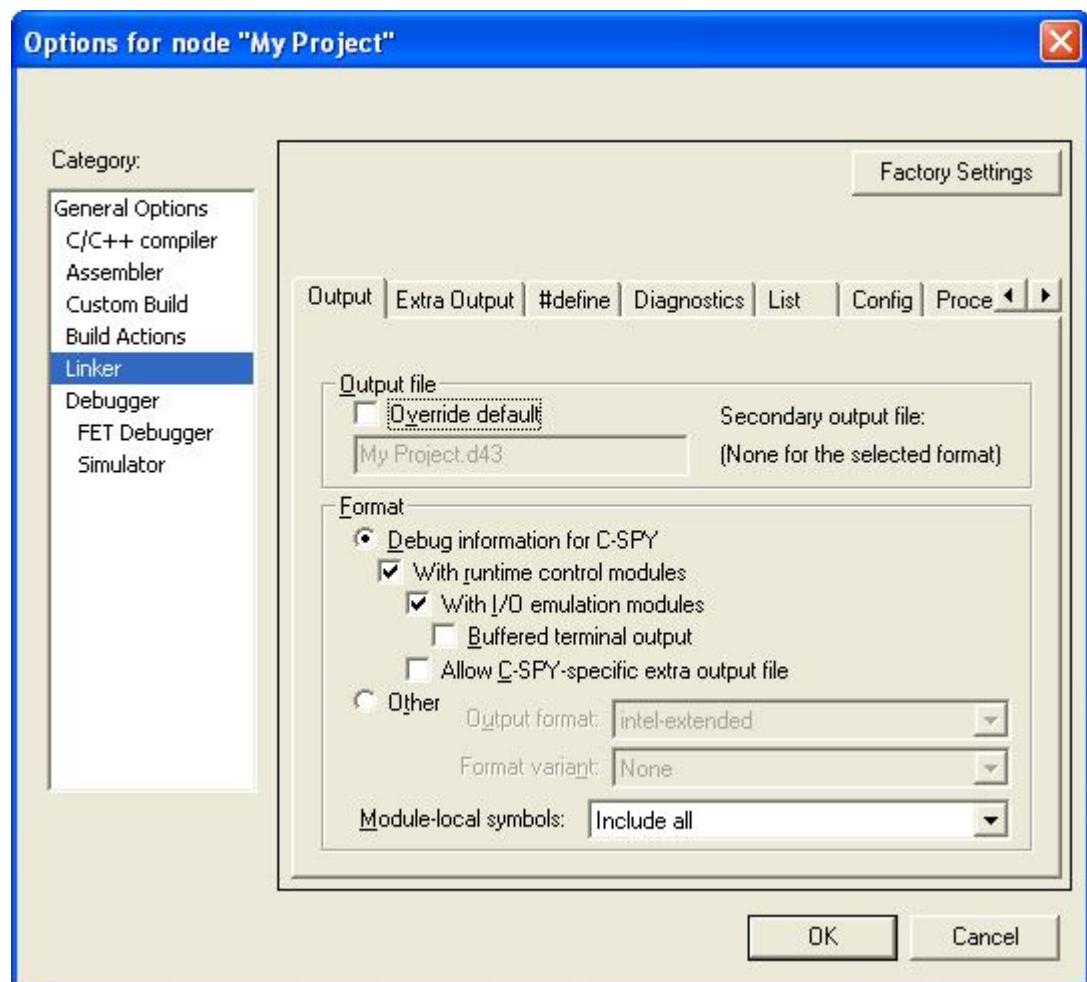
**Reset:** Bắt đầu thực thi lại từ địa chỉ 0x00.

Khi hoàn thành xong tất cả ta không thấy file \*hex tạo ra như các công cụ biên dịch khác. Tai phải làm thế nào? Nếu chúng ta để ý thì tại thư mục lưu lại dự án sẽ có file *My Project.d43*, tại sao tôi nói đến file này? Vì khi chúng ta biên dịch thành công và chọn Debug dự án thì phải có file thực thi tạo ra như file có đuôi \*hex chẳng hạn.



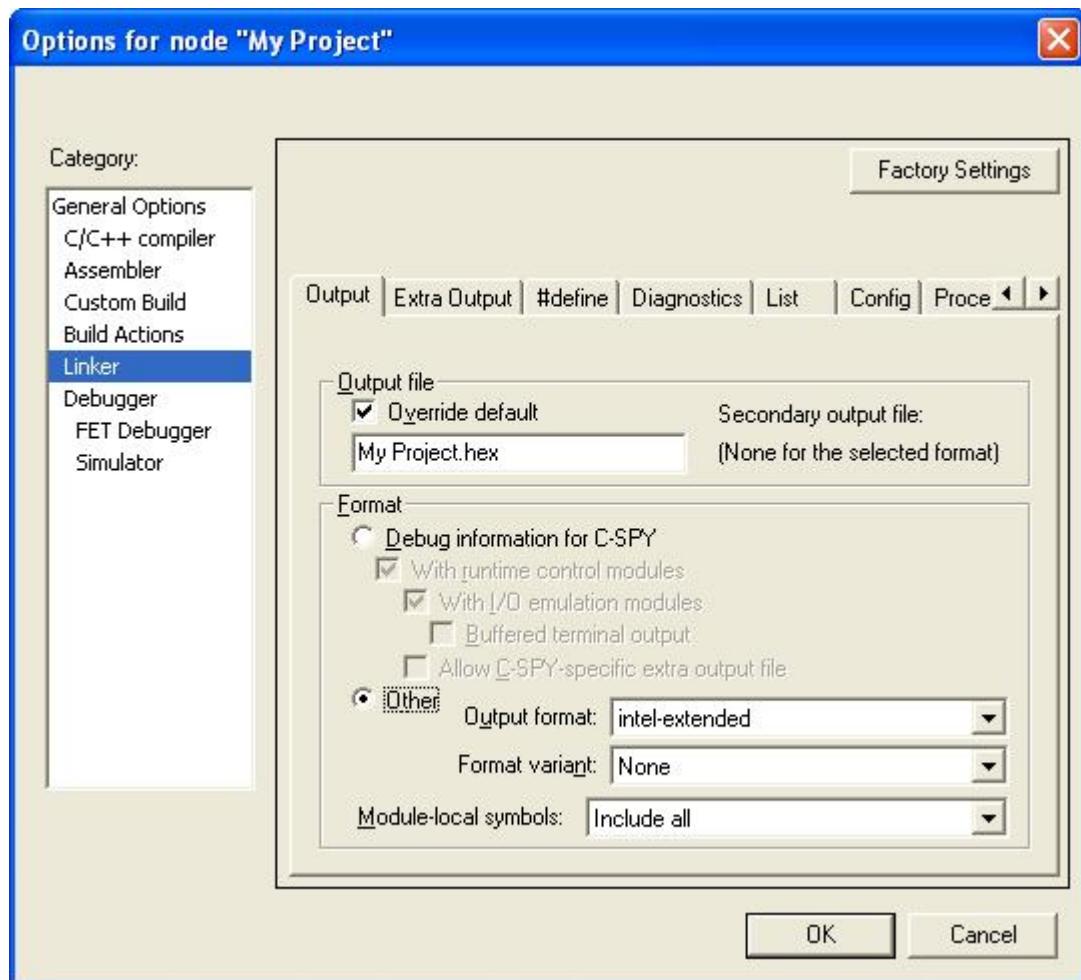
Hình 3-12 Một file thực thi được tạo ra.

Nhấp chuột phải vào *My Project - Debug* rồi chọn *Options...* Ta chọn *Linker* chọn thẻ *Output* ta chú ý đến *Output file* và *Format*. Ta thấy tại *Output file* có dạng file *My Project.d43*, vậy ta phải làm thế nào?



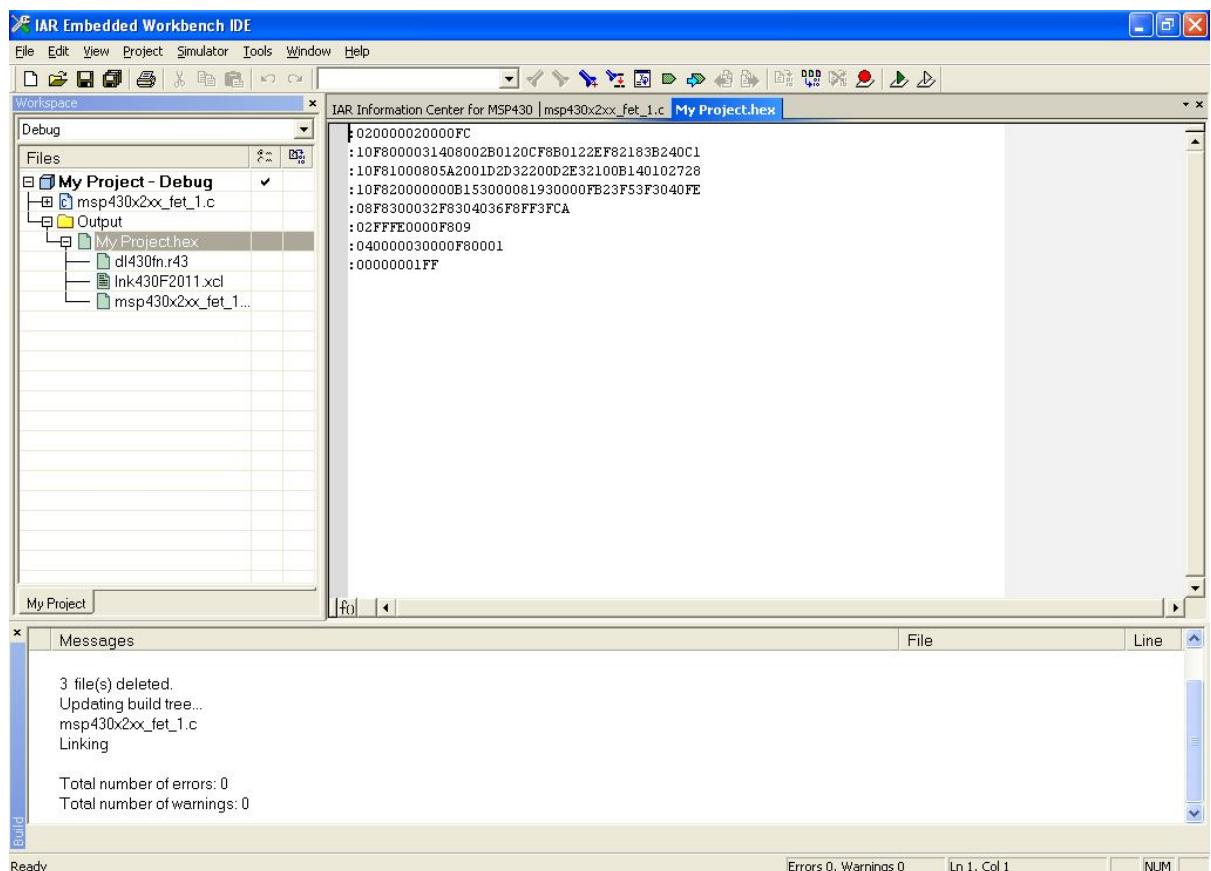
Hình 3-13 Cửa sổ Options....

Các bạn chú ý lại các thông số mà tôi đã hiệu chỉnh lại ở hình bên dưới nhé!  
Bây giờ chúng ta chọn *OK* và biên dịch lại là đã có file \*hex.



Hình 3-14 Cửa sổ Options....

Và kết quả sẽ như thế nào? Chúng ta xem hình 3-15



Hình 3-15 Tạo xong file \*.hex.

Tóm lại, các nghiên cứu trên đây chỉ là ở mức rất cơ bản. Ở phần mềm biên dịch này có rất nhiều công cụ khác mà tôi chưa nghiên cứu. Ngoài ra, còn có rất nhiều phần mềm khác có thể biên dịch cho họ MSP430 như: Code Composer Essentials, Rowley Associates CrossWorks for MSP430... Ở chương tiếp theo tôi xin trình bày về công cụ biên dịch Code Composer Essentials để chúng ta so sánh.

## CHƯƠNG IV: TRÌNH BIÊN DỊCH CODE COMPOSER ESSENTIALS



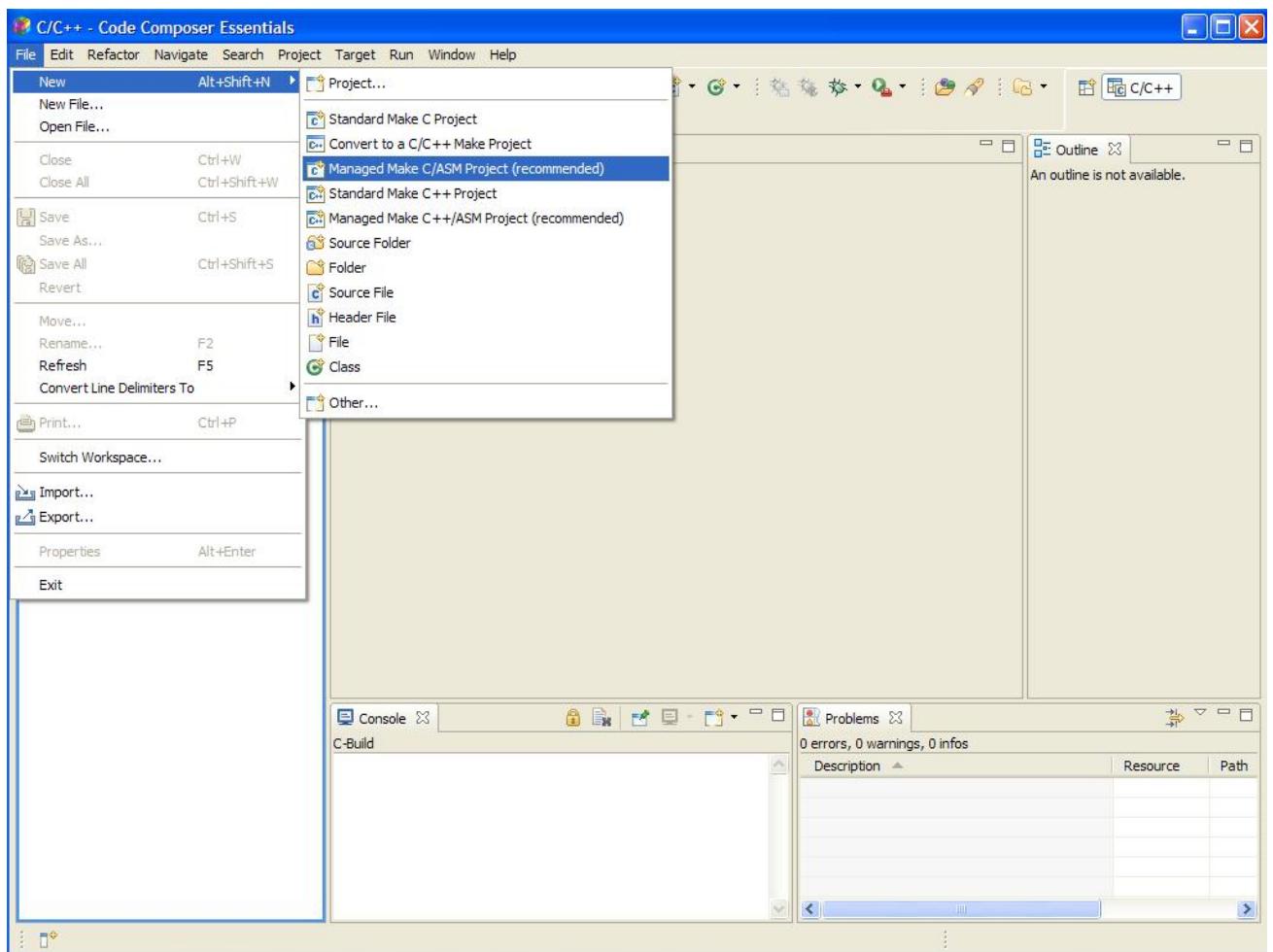
### 4. 1 Sơ lược

Code Composer Essentials có 3 phiên bản: Kickstart Version – Free và Professional Version - \$250. Bản miễn phí bị giới hạn 8KB code C, được hỗ trợ bởi TI.

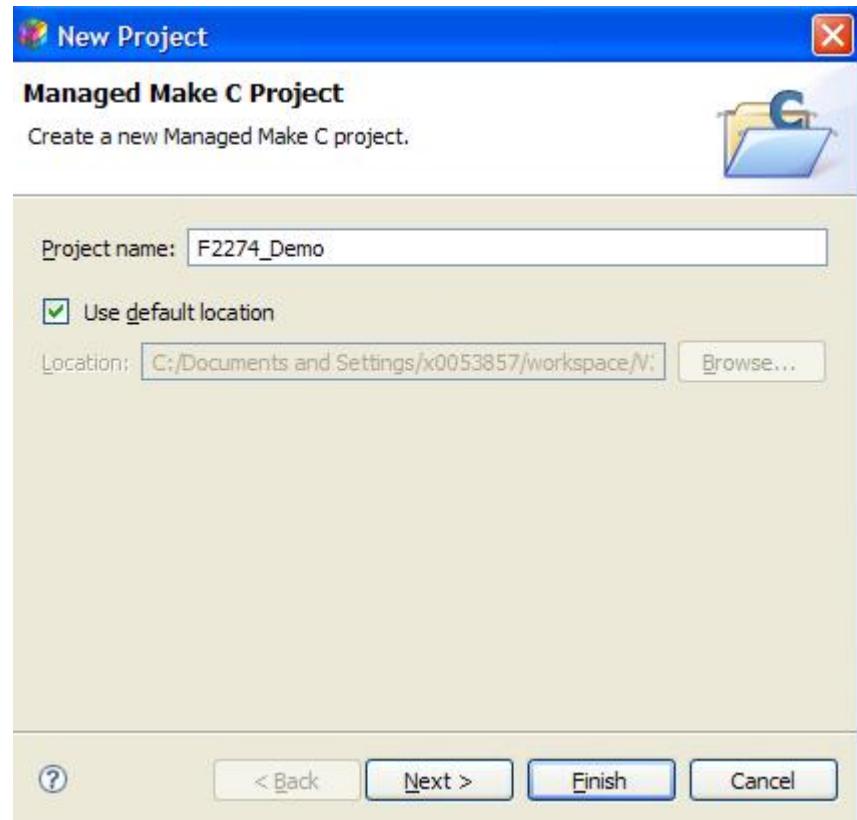


### 4. 2 Tạo dự án, biên dịch và mô phỏng

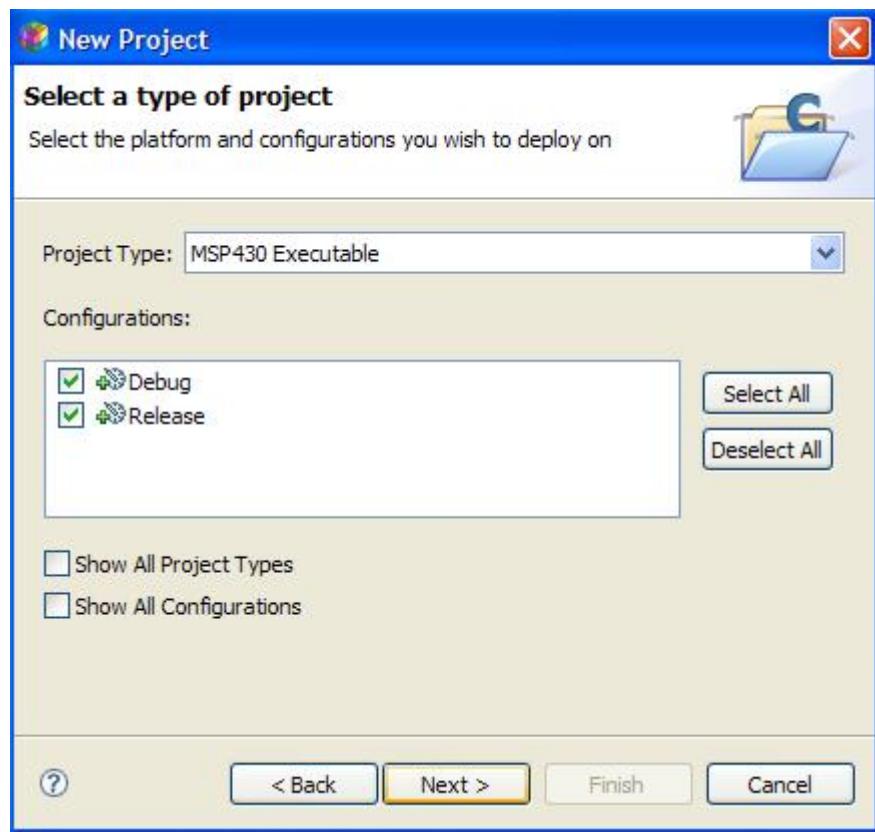
1. Open Code Composer Essentials. Under the "File" menu, choose New -> Managed Make C/ASM Project.



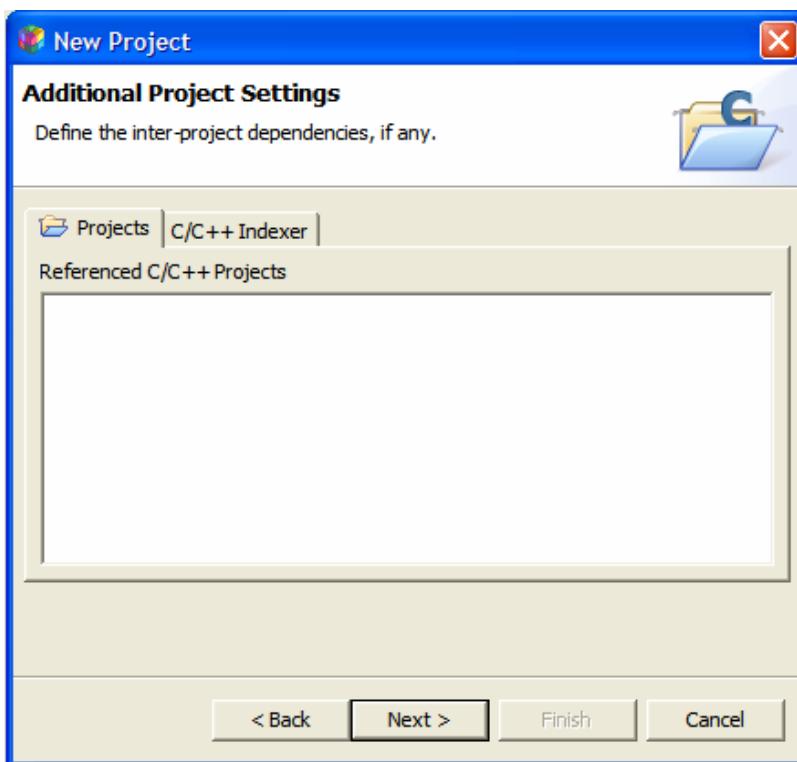
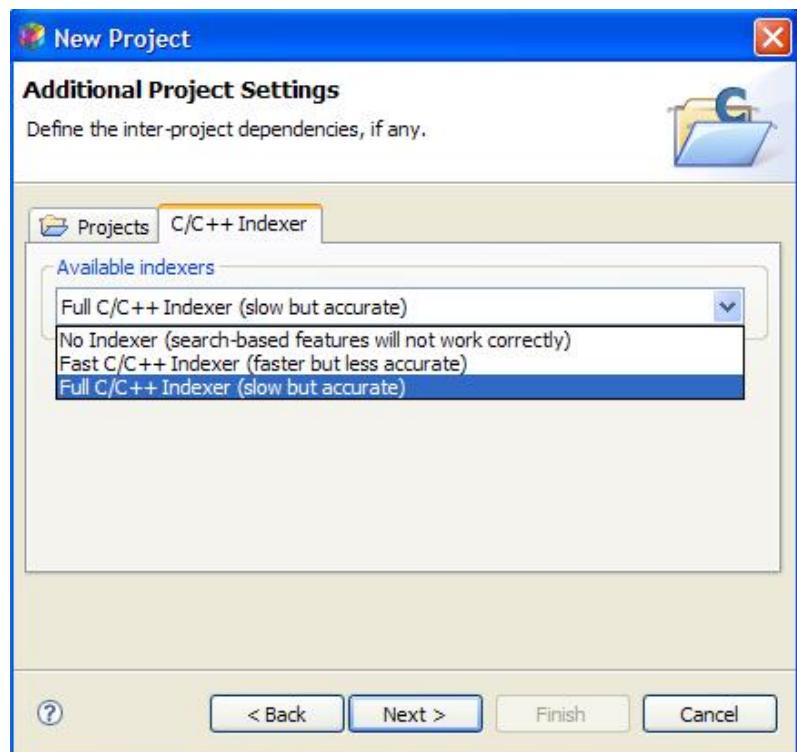
2. Then, when prompted, please name your project. Please be careful to use different names for different projects as one workspace is shared for all CCE projects. After naming click "Next."



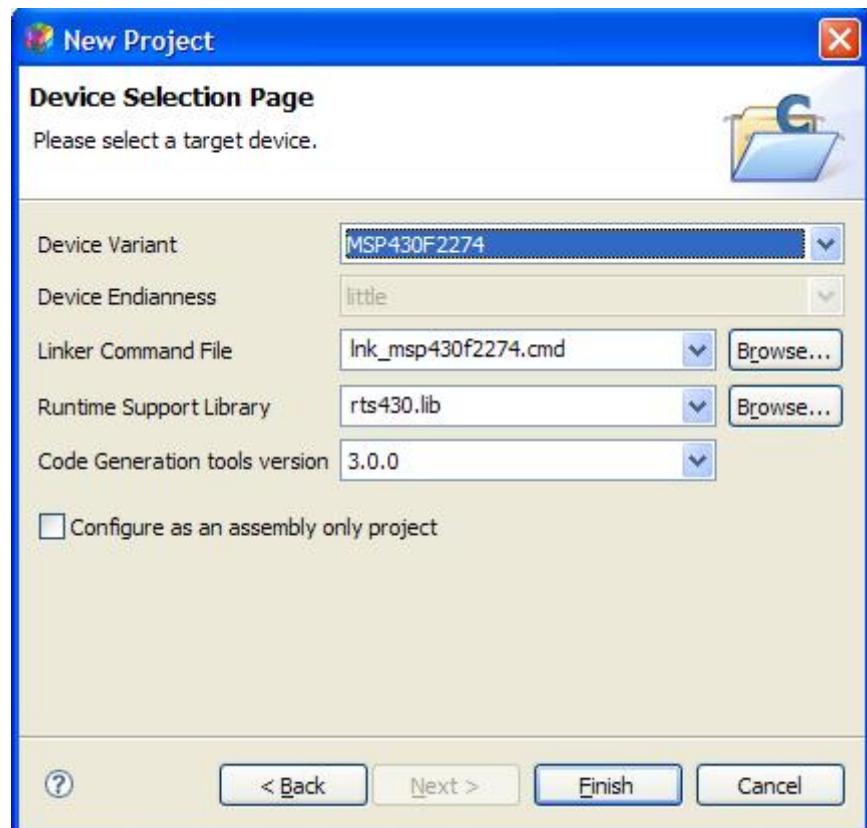
3. Please make sure that the Project Type is set to "MSP430 Executable" and that both Debug and Release configurations are checked.



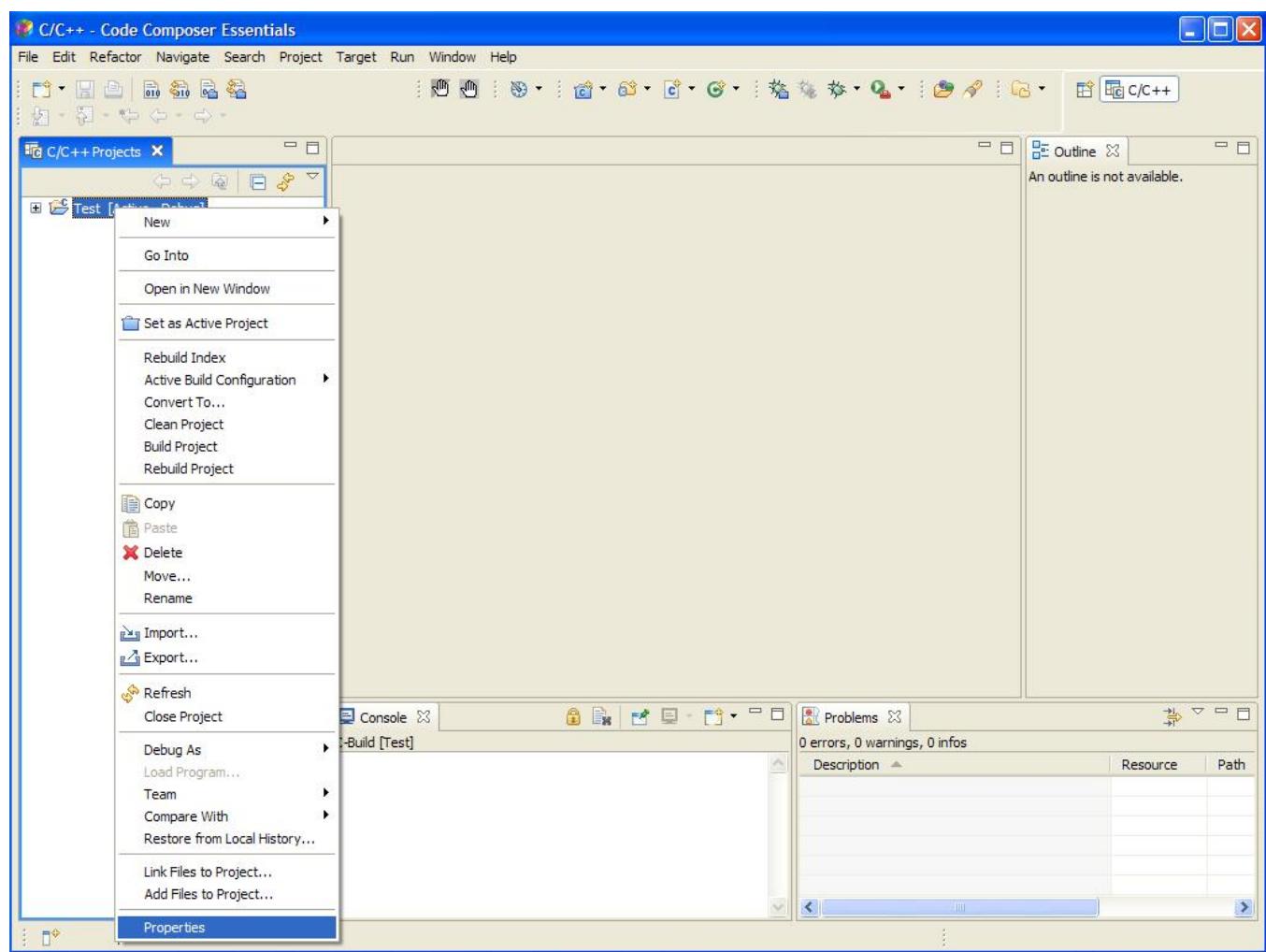
4. Please make sure no C/C++ Projects are referenced under the Projects tab. The C/C++ Indexer tab should reflect "Full C/C++ Indexer (slow but accurate)" as the indexers available. Once this is verified click "Next."



5. On the Device Selection Page, please choose the appropriate MSP430 under "Device Variant." Once chosen the correct Linker Command File should automatically appear. If you are planning on making an assembly-code project, please check the box "Configure as an assembly only project." If you are going to program in C/C++ leave this box unchecked. Click "Finish."



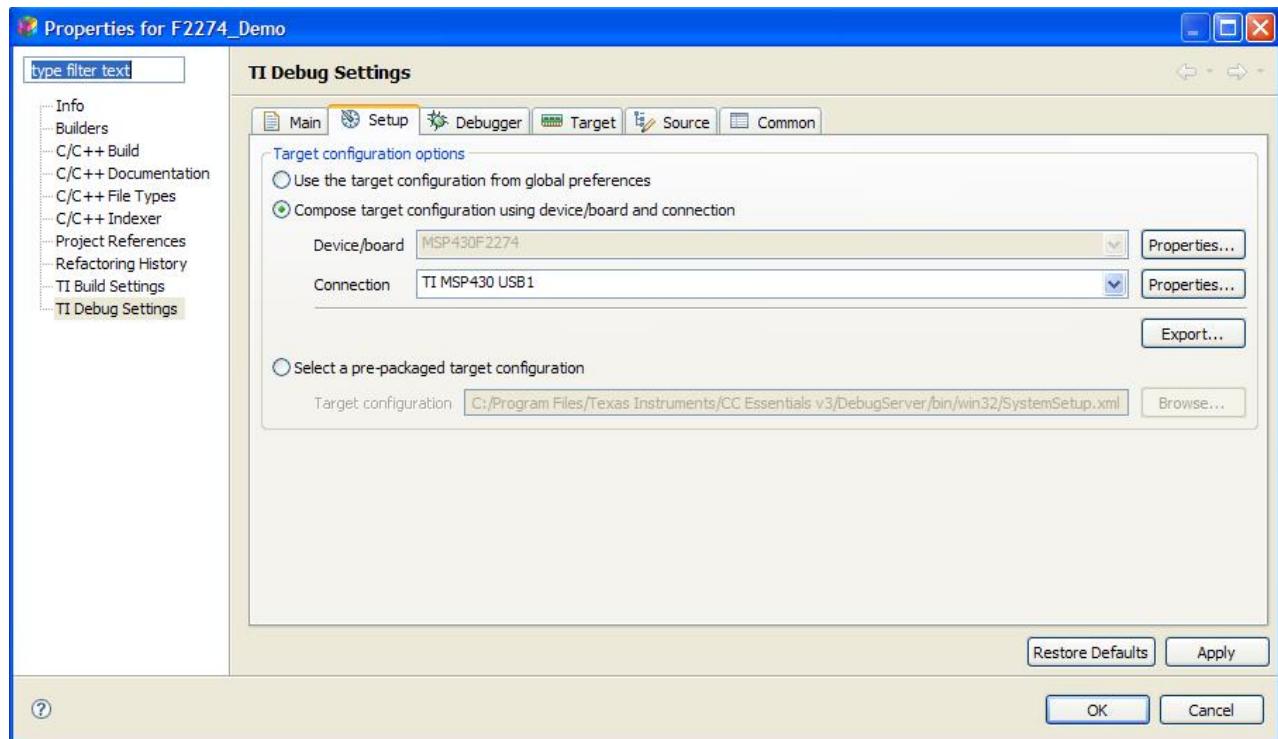
6. Now the project should appear at the left-hand side of the application under the C/C++ Projects tab. Right-click on your project and choose "Properties."



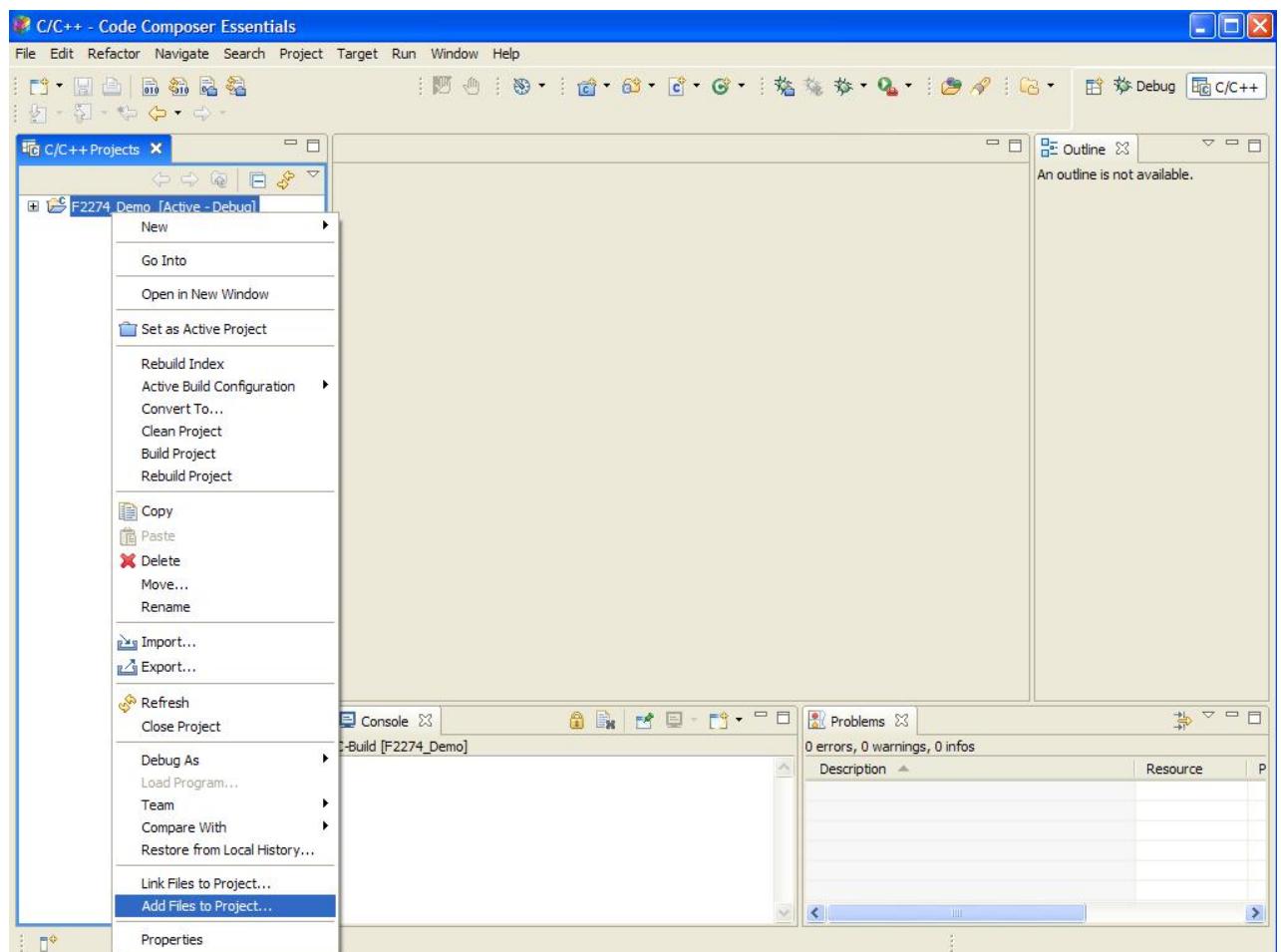
6. Please choose the TI Debug Settings category and choose the Setup tab. Pick the appropriate connection using the drop-down box as shown below. If you are using the EZ430 or the MSP-FET430UIF this should be configured by default.

Note: The JTAG protocol should be configured automatically, so only the connection needs to be specified.

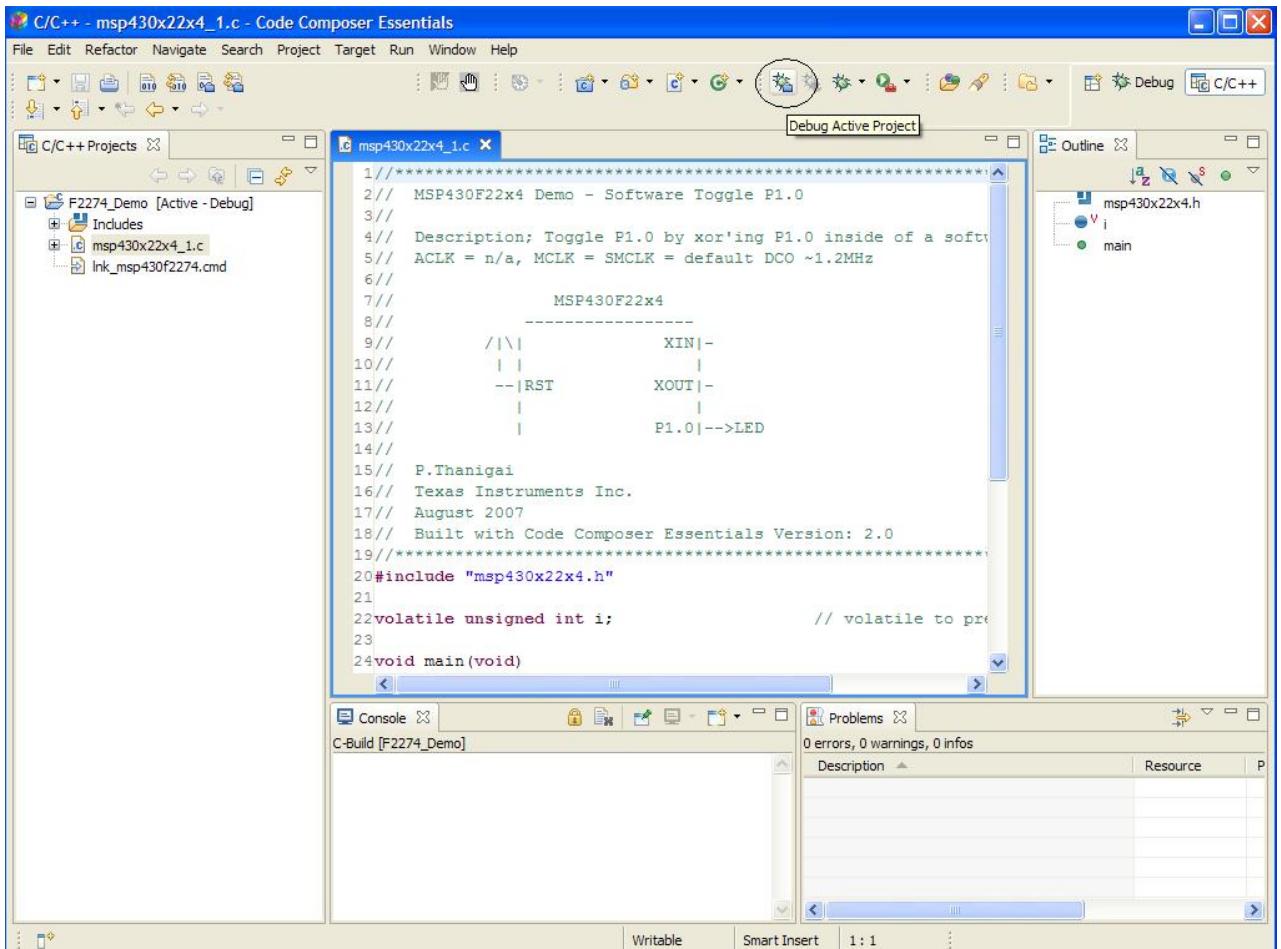
Click "OK" once these changes are made.



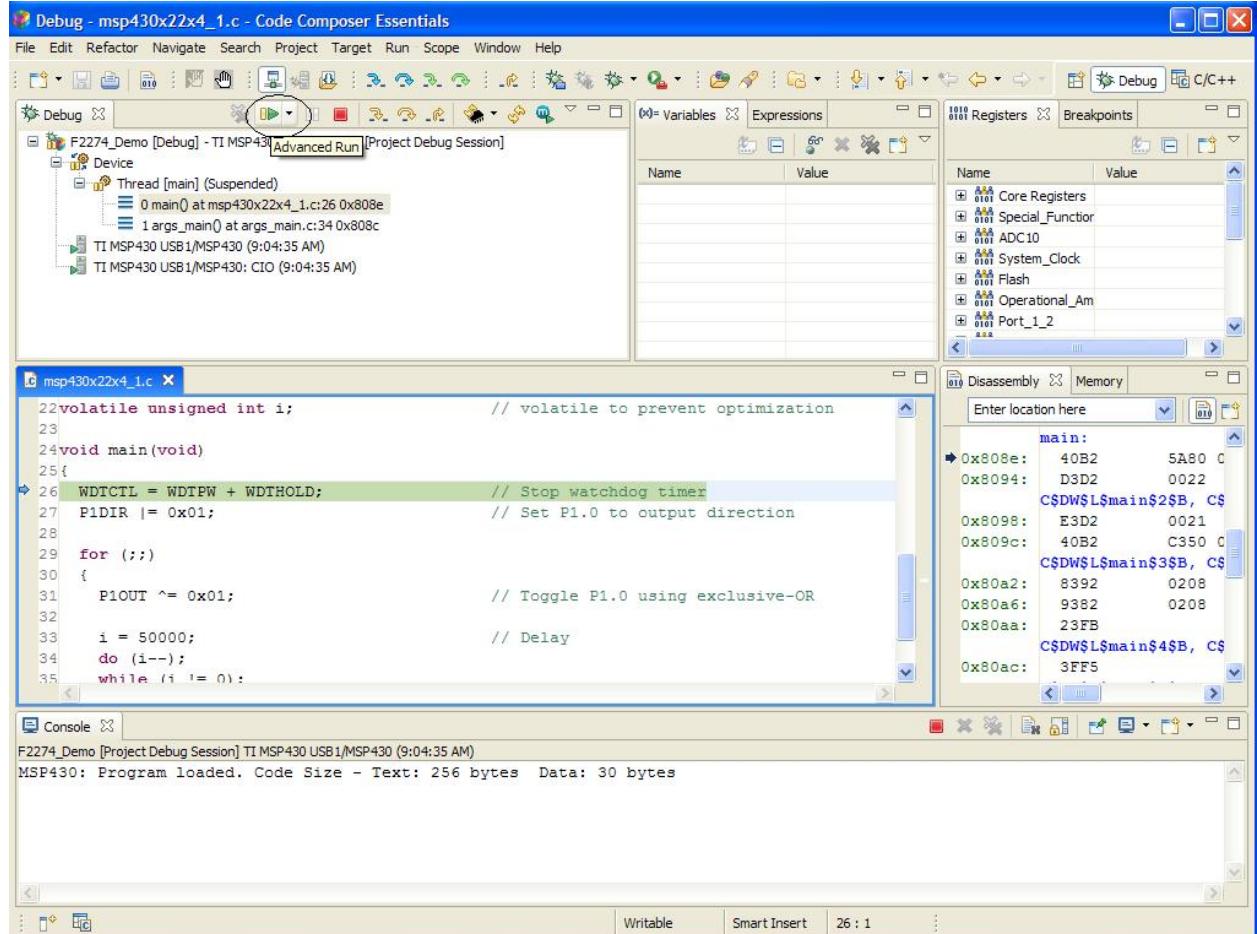
7. Now we are ready to add our code (of file type ".c" or ".asm") to our project. Right-click on your project once more and choose "Add files to project." Note: CCE V3 has default settings that are compatible with IAR syntax. To use the Version 2 CCE syntax please follow the steps located in section 2.1.3 in SLAU157:  
<http://www.ti.com/litv/pdf/slau157f>



8. Once this file is added, double click on it under the project tree to view the code in the main window of the C/C++ perspective. When you are ready to load your code onto the MSP430, choose "Debug Active Project" under the "Run" menu. Alternatively you can choose the Debug Active Project button at the upper-left side of the screen, as shown below.



9. Once your code successfully builds and loads onto your device, Code Composer will change the view to the debug perspective. At this point the code is completely loaded onto your device and is ready to run. To run this code, press F8 or choose run under the "Run" menu. Alternatively you can click the run button on the upper-left side of the screen as shown below.



### Further Reading:

CCE FET User's Guide: <http://www.ti.com/litv/pdf/slau157f>  
This Guide Discusses:

- Using CCE to Communicate with TI hardware tools
- Migrating from IAR to CCE

CCE Compiler's User's Guide:

<http://focus.ti.com/general/docs/techdocsabstract.tsp?abstractName=slau132b>

This Guide explains how to use these compiler tools:

- The Compiler
- Library-build process

- C++ name demangler

CCE Assembler User's Guide:

<http://focus.ti.com/general/docs/techdocsabstract.tsp?abstractName=slau131b>

The MSP430 Assembly Language Tools User's Guide explains how to use these assembly language tools:

- Assembler
- Archiver
- Linker
- Absolute Linker
- Cross-reference lister
- Disassembler
- Object file display utility
- Name utility
- Strip utility
- Hex conversion utility